

C语言

程序设计教程

基于Visual C++6.0环境

汤承林 姜仲秋 / 主编 张洪斌 / 主审

CYUYAN CHENGXU SHEJI JIAOCHENG



天津大学出版社
TIANJIN UNIVERSITY PRESS

C 语言程序设计教程

——基于 Visual C++ 6.0 环境

主 编 汤承林 姜仲秋
参 编 葛素娟 冯 钢
主 审 张洪斌



天津大学出版社
TIANJIN UNIVERSITY PRESS

内容提要

本书紧贴全国计算机等级考试(二级 C)的考核内容,较全面地讲述了 C 语言程序设计的基本知识,主要包括概述、数据类型、运算符及表达式,基本输入/输出函数,选择结构,循环结构,数组,函数,指针,编译预处理命令,结构与共用,文件和图书管理系统案例介绍等,书后安排有与各章内容相关的 15 个实验。

本书章节例题有解题思路、参考代码、程序运行结果图、程序说明和注意点等。每一章都精选与全国计算机等级考试(二级 C)难度匹配的练习题,有助于读者复习、巩固所学知识。

本书结构严谨、重点突出、由浅入深、举例实用。本书的编程环境采用 Visual C++6.0,所有参考代码都在 Visual C++6.0 上调试通过。

本书可以作为各类高职高专、高等院校计算机专业及非计算机专业学习“计算机程序设计”的课程教材,也可作为参加全国计算机等级考试(二级 C)和广大计算机爱好者学习 C 语言程序设计的参考书。

图书在版编目(CIP)数据

C 语言程序设计教程:基于 Visual C++6.0 环境/汤承林,姜仲秋主编. —天津:天津大学出版社,2009.8

ISBN 978-7-5618-3192-2

I. C… II. ①汤…②姜… III. C 语言—程序设计—高等学校—教材 IV. TP312

中国版本图书馆 CIP 数据核字(2009)第 152435 号

出版发行 天津大学出版社

出版人 杨欢

地址 天津市卫津路 92 号天津大学内(邮编:300072)

电话 发行部:022-27403647 邮购部:022-27402742

网址 www.tjup.com

印刷 昌黎太阳红彩色印刷有限责任公司

经销 全国各地新华书店

开本 185 mm × 260 mm

印张 22

字数 550 千

版次 2009 年 8 月第 1 版

印次 2009 年 8 月第 1 次

定价 36.00 元

凡购本书,如有缺页、倒页、脱页等质量问题,烦请向我社发行部门联系调换

版权所有 侵权必究

前 言

C 语言是国内外目前广泛应用的程序设计语言。它功能强大,数据类型丰富,使用灵活,通用性强,并兼有面向硬件编程的低级语言特性和可读性强的高级语言特性。因此,C 语言已成为高校计算机专业和非计算机专业学生必修的一门计算机语言。

然而,在 C 语言的教学过程中,预期教学目标与最终教学效果往往有明显的差距,教师感觉难教,学生感觉难学、难理解,学了也不会编程。特别是高职高专院校的学生,学习时若过分注重选择适合高职高专学生学习特点的实用性 C 语言教材,学习后参加全国计算机等级考试(二级 C)时通过率很低;若学习时过分注重选择适合本科院校学生特点的学科性很强的 C 语言教材,理论性、学科性又太强,注重数据结构知识的讲解,这类教材适合参加全国计算机等级考试,但实用性差。在现时高职高专以“实用、够用”为度的情况下,教材以注重实用性为前提,必须改革教材内容,寻找能兼顾两者,既实用又具学科性的 C 语言教材。本教材就是在这种背景下,由具有多年高职高专 C 语言教学和开发经验的教师,经过认真分析高职高专学生学习特点和全国计算机等级考试(二级 C)的要求而编写的。

在本书编写过程中,主要从如下几个方面考虑。

(1)充分考虑全国计算机等级考试要求,教材内容采用全国计算机等级考试(二级 C)考核内容。

(2)教学内容的选择注重实用,选择易于被高职高专学生接受的例子。每个例子基本上分为解题思路、参考代码、解题说明、程序运行结果图和注意点。

(3)以一个“图书管理系统”案例贯穿全书,以期使学生学习 C 语言后,能知道 C 语言到底能做什么,如何编写实用的 C 语言应用程序。“图书管理系统”案例根据需要分解在本书的部分章节中,针对章节所学内容,改造成利用所学章节内容编写的“图书管理系统”子模块。

(4)充分考虑学生学习资料少、练习少、无适合全国等级考试练习题可做的窘境,在每章后均附有大量全国计算机等级考试练习题。习题主要分为选择题、填空题和编程题三类。

(5)书后安排了 15 个实验,供学生上机练习。实验内容充分考虑到高职高专学生特点,把每个实验内容分为教师指导、练习、作业三部分。

本书的编程环境采用 Visual C++6.0,所有参考代码都在 Visual C++6.0 上调试通过。

全书由江苏淮安信息职业技术学院汤承林、姜仲秋副教授任主编,张洪斌研究员任主审。本书编写分工如下:第 3、4、6 和 10 章由姜仲秋编写;第 2、7、9 和 12 章由汤承林编写;第 5 和 8 章由葛素娟编写;第 1 和 11 章及实验部分由冯钢编写。本书在编写过程中得到了管曙亮高级工程师等人的帮助和支持,在此表示衷心感谢!

由于作者水平有限,加上时间仓促,错误之处在所难免,恳请广大读者批评指正。联系信箱:TCL@hreit.edu.cn 或 TCL12345678900@163.com

编 者
2009.5

目 录

第1章 概述	1
1.1 程序与算法	1
1.1.1 程序	1
1.1.2 算法	1
1.2 C语言的特点、源程序的书写格式	3
1.2.1 C语言的特点	3
1.2.2 C程序的构成	4
1.2.3 C语言程序的书写格式	4
1.2.4 C程序的三种基本结构和流程图	5
1.3 案例简介	6
1.4 Visual C++ 6.0 上机操作	6
1.4.1 C程序编译与可执行文件的生成	6
1.4.2 Visual C++ 6.0 的上机操作步骤	7
1.4.3 C程序调试	10
本章小结	13
习题一	13
第2章 数据类型、运算符及表达式	16
2.1 基本数据类型	16
2.2 常量与变量	17
2.2.1 常量与变量定义	17
2.2.2 整型数据	19
2.2.3 实型数据	22
2.2.4 字符型数据	23
2.2.5 字符串常量	25
2.3 数据类型转换	27
2.4 运算符及表达式	27
2.4.1 算术运算符与表达式	28
2.4.2 赋值运算符与表达式	30
2.4.3 逗号运算符	32
本章小结	32
习题二	33
第3章 基本输入/输出函数	37
3.1 输入/输出函数	37
3.1.1 格式化输入函数 scanf()	37

3.1.2 格式化输出函数 printf()	38
3.2 字符输入/输出函数	40
3.2.1 字符输入函数 getchar()	40
3.2.2 字符输出函数 putchar()	40
3.3 案例应用举例	41
本章小结	42
习题三	42
第4章 选择结构	48
4.1 关系运算和逻辑运算	48
4.1.1 关系运算符与表达式	48
4.1.2 逻辑运算符与表达式	49
4.2 if 语句	50
4.2.1 简单 if 语句	50
4.2.2 二分支 if 语句	51
4.2.3 二分支 if 语句嵌套	52
4.2.4 多分支 if 语句	53
4.2.5 条件运算符	55
4.3 switch 语句	55
4.4 综合实例	58
4.5 案例应用举例	62
本章小结	63
习题四	63
第5章 循环结构	70
5.1 goto 语句	70
5.2 for 语句	71
5.2.1 for 循环语句	71
5.2.2 for 循环嵌套	73
5.3 while 和 do-while 循环语句	76
5.3.1 while 循环语句	76
5.3.2 do-while 循环语句	78
5.3.3 for、while 和 do-while 循环嵌套	79
5.4 break 语句和 continue 语句	81
5.4.1 break 语句	81
5.4.2 continue 语句	82
5.5 综合实例	83
5.6 案例应用举例	85
本章小结	86
习题五	87
第6章 数组	98

6.1 一维数组	98
6.1.1 一维数组的定义及初始化	99
6.1.2 一维数组元素的引用	100
6.2 二维数组	105
6.2.1 二维数组的定义	105
6.2.2 二维数组的初始化	106
6.2.3 二维数组元素的引用	106
6.2.4 多维数组的定义	108
6.3 字符数组	109
6.3.1 字符数组的定义	109
6.3.2 字符数组的初始化	109
6.3.3 字符串与字符数组	110
6.3.4 字符串的输入/输出	111
6.3.5 字符串输入/输出函数	111
6.3.6 常用字符串处理函数	112
6.4 综合实例	114
6.5 案例应用举例	116
本章小结	118
习题六	118
第 7 章 函数	130
7.1 函数定义	130
7.2 函数的参数和函数的值	132
7.2.1 形式参数和实际参数	132
7.2.2 函数的返回值	133
7.2.3 函数的调用	134
7.2.4 函数原型	135
7.2.5 函数的调用实例	135
7.3 函数的嵌套调用	138
7.4 函数的递归调用	140
7.5 数组作为函数参数	141
7.5.1 数组元素作函数实参	142
7.5.2 数组名作为函数参数	143
7.6 变量的存储类别、作用域和生存期	146
7.6.1 局部变量与全局变量	146
7.6.2 变量的存储类别	149
7.7 内部函数与外部函数	152
7.7.1 内部函数	152
7.7.2 外部函数	152
7.8 案例应用举例	153

8.0 本章小结	154
8.0 习题七	154
第 8 章 指针	167
8.1 指针的概念和定义	167
8.1.1 地址和指针的概念	167
8.1.2 指针变量的定义	168
8.1.3 指针的操作	169
8.1.4 指向指针的指针	172
8.2 指针变量与函数参数	174
8.2.1 指针作为函数参数	174
8.2.2 值传递与地址传递的区别与联系	176
8.3 指针与数组	177
8.3.1 指向数组的指针	177
8.3.2 数组指针访问数组	178
8.3.3 指针变量访问数组	179
8.3.4 数组名作函数参数	180
8.4 指针与二维数组	182
8.4.1 二维数组的地址	182
8.4.2 指向由 n 个元素组成的一维数组的指针变量	184
8.4.3 指向二维数组元素的指针变量	186
8.5 字符串与指针	187
8.5.1 字符串的表示形式	187
8.5.2 字符数组与字符串指针作函数参数	188
8.6 指针数组	190
8.7 指针与函数	193
8.7.1 函数指针	193
8.7.2 返回值是指针的函数	195
8.8 main() 函数的参数	196
8.9 案例应用举例	197
8.0 本章小结	198
8.0 习题八	199
第 9 章 编译预处理命令	207
9.1 宏定义	207
9.1.1 不带参数的宏定义	207
9.1.2 带参数的宏定义	209
9.2 文件包含	210
9.3 条件编译	212
9.0 本章小结	215
9.0 习题九	215

第10章 结构与共用	219
10.1 结构与结构变量	219
10.1.1 结构的定义	219
10.1.2 结构变量的定义	220
10.1.3 结构变量的引用与赋值	222
10.1.4 结构变量的初始化	223
10.2 结构数组	224
10.2.1 结构数组的定义	224
10.2.2 结构数组的初始化	224
10.2.3 结构数组应用	225
10.3 指向结构的指针变量	226
10.3.1 指向结构变量的指针	226
10.3.2 指向结构数组的指针	228
10.3.3 指向结构变量的指针变量作函数参数	229
10.3.4 指向结构数组的指针变量作函数参数	230
10.4 类型定义符 typedef	232
10.5 结构与链表	233
10.5.1 动态存储结构	233
10.5.2 链表操作	235
10.6 共用	243
10.7 位运算	247
本章小结	250
习题十	250
第11章 文件	259
11.1 文件概述	259
11.1.1 文件及分类	259
11.1.2 文件类型指针	260
11.1.3 文件的打开与关闭	260
11.2 文本文件的读写	262
11.2.1 字符输入/输出函数 fgetc() 和 fputc() 及文件结束检测函数 feof()	262
11.2.2 字符串读写函数 fgets() 和 fputs()	265
11.3 二进制文件读写	266
11.4 文件的随机读写	269
11.4.1 文件定位	269
11.4.2 文件的随机读写	270
本章小结	271
习题十一	271
第12章 图书管理系统案例介绍	277
12.1 案例的目的和任务	277

12.2	案例知识要点综述	277
12.3	案例实训要求	277
12.4	案例需求分析	277
12.5	案例总体设计	277
12.6	案例详细设计	279
12.7	案例总结	300
实验		301
实验一	熟悉 Visual C++ 6.0 环境	301
实验二	数据类型及运算符	302
实验三	输入/输出语句	303
实验四	选择结构	305
实验五	循环结构(一)	307
实验六	循环结构(二)	310
实验七	数组(一)	312
实验八	数组(二)	314
实验九	函数(一)	316
实验十	函数(二)	319
实验十一	指针(一)	322
实验十二	指针(二)	324
实验十三	结构与共用(一)	328
实验十四	结构与共用(二)	331
实验十五	文件	334
附录 1 ASCII 表		337
附录 2 C 库函数所在头文件		338
参考文献		339
11.1.1	类及类成员	11.1.1
11.1.2	类成员函数	11.1.2
11.1.3	类成员变量	11.1.3
11.2	类成员函数	11.2
11.3	类成员变量	11.3
11.4	类成员函数	11.4
11.5	类成员变量	11.5
11.6	类成员函数	11.6
11.7	类成员变量	11.7
11.8	类成员函数	11.8
11.9	类成员变量	11.9
11.10	类成员函数	11.10
11.11	类成员变量	11.11
11.12	类成员函数	11.12
11.13	类成员变量	11.13
11.14	类成员函数	11.14
11.15	类成员变量	11.15
11.16	类成员函数	11.16
11.17	类成员变量	11.17
11.18	类成员函数	11.18
11.19	类成员变量	11.19
11.20	类成员函数	11.20
11.21	类成员变量	11.21
11.22	类成员函数	11.22
11.23	类成员变量	11.23
11.24	类成员函数	11.24
11.25	类成员变量	11.25
11.26	类成员函数	11.26
11.27	类成员变量	11.27
11.28	类成员函数	11.28
11.29	类成员变量	11.29
11.30	类成员函数	11.30
11.31	类成员变量	11.31
11.32	类成员函数	11.32
11.33	类成员变量	11.33
11.34	类成员函数	11.34
11.35	类成员变量	11.35
11.36	类成员函数	11.36
11.37	类成员变量	11.37
11.38	类成员函数	11.38
11.39	类成员变量	11.39
11.40	类成员函数	11.40
11.41	类成员变量	11.41
11.42	类成员函数	11.42
11.43	类成员变量	11.43
11.44	类成员函数	11.44
11.45	类成员变量	11.45
11.46	类成员函数	11.46
11.47	类成员变量	11.47
11.48	类成员函数	11.48
11.49	类成员变量	11.49
11.50	类成员函数	11.50
11.51	类成员变量	11.51
11.52	类成员函数	11.52
11.53	类成员变量	11.53
11.54	类成员函数	11.54
11.55	类成员变量	11.55
11.56	类成员函数	11.56
11.57	类成员变量	11.57
11.58	类成员函数	11.58
11.59	类成员变量	11.59
11.60	类成员函数	11.60
11.61	类成员变量	11.61
11.62	类成员函数	11.62
11.63	类成员变量	11.63
11.64	类成员函数	11.64
11.65	类成员变量	11.65
11.66	类成员函数	11.66
11.67	类成员变量	11.67
11.68	类成员函数	11.68
11.69	类成员变量	11.69
11.70	类成员函数	11.70
11.71	类成员变量	11.71
11.72	类成员函数	11.72
11.73	类成员变量	11.73
11.74	类成员函数	11.74
11.75	类成员变量	11.75
11.76	类成员函数	11.76
11.77	类成员变量	11.77
11.78	类成员函数	11.78
11.79	类成员变量	11.79
11.80	类成员函数	11.80
11.81	类成员变量	11.81
11.82	类成员函数	11.82
11.83	类成员变量	11.83
11.84	类成员函数	11.84
11.85	类成员变量	11.85
11.86	类成员函数	11.86
11.87	类成员变量	11.87
11.88	类成员函数	11.88
11.89	类成员变量	11.89
11.90	类成员函数	11.90
11.91	类成员变量	11.91
11.92	类成员函数	11.92
11.93	类成员变量	11.93
11.94	类成员函数	11.94
11.95	类成员变量	11.95
11.96	类成员函数	11.96
11.97	类成员变量	11.97
11.98	类成员函数	11.98
11.99	类成员变量	11.99
12.00	类成员函数	12.00

第 1 章 概述

C 语言的设计始于 1971 年,后经多次改进,迅速成为最受欢迎的计算机编程语言之一。许多流行软件都是用 C 语言编写的。

C 语言的编程环境有 Turbo C、Borland C、Visual C++ 等,它们都内嵌了 C 语言编译程序。本章介绍 C 语言程序的基本知识:

- (1) 算法、C 语言的特点、C 语言的构成、书写格式介绍;
- (2) 本书案例“图书管理系统”简介;
- (3) Visual C++6.0 上机操作步骤介绍。

1.1 程序与算法

1.1.1 程序

“程序”一词来自生活,通常指完成某件事情的一种既定的方式和过程。可以把程序看成对一系列动作的执行过程的描述。例如,学生去食堂吃饭的行为可以描述为:

- (1) 带上饭碗去食堂;
- (2) 排队;
- (3) 打饭;
- (4) 刷卡;
- (5) 就餐;
- (6) 洗碗;
- (7) 离开食堂。

这是个生活中的程序性活动的例子,计算机完成某件事情与生活中解决某个问题的过程非常相似。为了完成人们交给计算机的问题,计算机提供了一套指令,其中每一条指令对应着计算机能执行的一个基本动作,为了让计算机解决某个问题而编写的逐条执行的指令序列就称为程序。

1.1.2 算法

为解决一个问题而采取的方法与步骤,称为“算法”(algorithm)。或者说,算法是解题方法的精确描述。解决一个问题的过程就是实现一个算法的过程。对同一个问题,往往有不同的解题方法,各种方法的执行速度可能不同;为了有效地执行程序,应当选择适当的算法。选择算法的主要标准是考虑它的正确性、可靠性、简单性和易理解性,其次是考虑所需要的存储空间和执行速度等。

一个程序应包括对数据的描述和对操作的描述。

- (1) 对数据的描述。在程序中要指定数据的类型和组织形式,即数据结构(data struc-

ture)。

(2)对操作的描述。即操作步骤,也就是算法。著名计算机科学家 Nikiklaus Wirth 对程序提出如下公式:

程序 = 数据结构 + 算法

一个算法应具有以下特点。

- (1)有穷性。一个算法应包含有限的操作步骤,而不能是无限的。
- (2)确定性。算法中每一个步骤应当是确定的,而不能是含糊的、模棱两可的。
- (3)有零个或多个输入。所谓输入是指执行指定的算法时,需要外界提供输入信息。
- (4)有一个或多个输出。
- (5)有效性。算法中每一个步骤应当能有效地执行,并得到确定的结果。

对于程序设计人员,必须会设计算法,并根据算法写出程序。

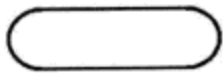
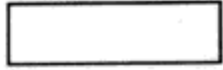
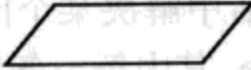



【例 1.1】 求 $1 + 2 + 3 + \cdots + 100$ 。假设 SUM 代表累加和,N 代表累加数个数。其程序如下。

- (1)SUM = 0,N = 1。
- (2)求 SUM + N,得到的结果放在 SUM 中。
- (3)N 加 1 后放在 N 中。
- (4)如果 $N \leq 100$,跳转到(2);如果 $N > 100$,则算法结束。此时 SUM 的值就是所求结果。

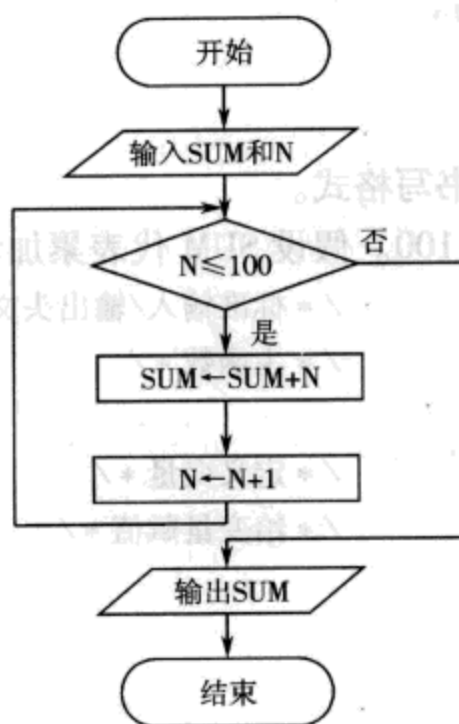
算法通常按某种规则来描述,常用的有自然语言、传统流程图、N-S 结构流程等。

流程图是算法的一种图形化表示方式,它直观、清晰,有利于人们设计算法。传统流程图常用的符号如表 1.1 所示。

表 1.1 传统流程图常用符号

符号	描述
	程序开始或结束
	计算机步骤/处理符号
	输入/输出指令
	判断和分支
	连接符
	流程线

前面给出的计算 1 到 100 之和的算法用流程图表示如图 1.1 所示。

图 1.1 $1 + 2 + \dots + 100$ 流程图

1.2 C 语言的特点、源程序的书写格式

1.2.1 C 语言的特点

C 语言是现今最流行的几种程序设计语言之一。与其他语言相比, C 语言具有如下主要特点。

(1) 语言简洁、紧凑, 使用方便、灵活。ANSI C 一共只有 32 个关键字, 如表 1.2 所示。

表 1.2 ANSI C 的 32 个关键字

auto	break	case	char	const	continue	default
do	double	else	enum	extern	float	for
goto	if	int	long	register	return	short
signed	static	sizeof	struct	switch	typedef	union
unsigned	void	volatile	while			

(2) 运算符丰富。

(3) 数据结构丰富, 具有现代化语言的各种数据结构。

(4) 具有结构化的控制语句。

(5) 语法限制不太严格, 程序设计自由度大。

(6) 允许直接对内存物理地址的数据进行操作, 可以实现汇编语言的大部分功能, 同时具有高级语言的结构化和可读性、可移植性等优点。有人把 C 语言称为“高级语言中的低级语言”或“中级语言”, 但一般仍习惯将 C 语言称为高级语言, 因为 C 程序也要通过编译、连接才能得到可执行的目标程序。

(7)生成目标代码运行效率高。

1.2.2 C 程序的构成

例 1.2 展示 C 程序的构成与书写格式。

【例 1.2】求 $1+2+3+\cdots+100$ 。假设 SUM 代表累加和, N 代表累加和个数。

```
#include <stdio.h>          /* 标准输入/输出头文件 */
void main()                 /* 主函数 */
{
    int SUM, N;              /* 定义变量 */
    SUM = 0;                 /* 给变量赋值 */
    N = 1;
    while( N <= 100)
    {
        SUM = SUM + N;
        N = N + 1;
    }
    printf("SUM = %d\n", SUM); //输出结果
}
```

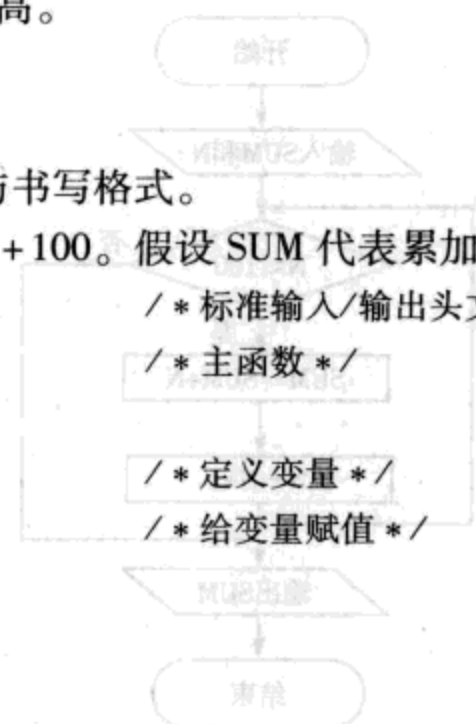


图 1.2 求 $1+2+3+\cdots+100$ 的 C 程序

程序分析如下。

(1)所有 C 语言程序必须包括一个函数 `main()`,表示主函数。它是 C 语言程序的入口。函数 `main()` 后面由一对花括号 `{ }` 括起来的部分称为程序的主体,程序从函数 `main()` 的第 1 个可执行语句开始执行。

(2)`/* */`和`//.....`表示注释部分,用于对程序进行说明,增强代码的可读性,但对程序编译与运行不起作用。前者表示块注释,后者表示单行注释。语句的注释可以有多行,可以放在程序的任何地方,但不可插入到关键字或标识符中间,否则编译时会出错。

(3)`printf()`是 C 语言程序中标准库函数的输出函数,其功能是输出双引号内的字符串,而“`\n`”是换行控制符,即输出完毕后自动换行。由于 `printf()`是 C 语言程序中的库函数,所以在程序的开始必须引用库函数的接口说明文件 `stdio.h`,此文件中包含函数 `printf()` 的原型说明。

(4)主函数前的关键字 `void` 表示该函数不返回任何结果,缺省时表示返回整型变量值。

1.2.3 C 语言程序的书写格式

根据例 1.2,可以总结 C 语言程序的书写格式如下。

(1)C 程序采用块注释。块注释书写格式为:

```
/* 注释部分 */
```

注释只是增强程序的可读性,不参加程序的编译与执行。书写时要注意“/”与“*”之间不能含有空格。C 程序在 Visual C++6.0 编程环境中,也可以采用 C++ 的注释方法,即要对某行进行注释,只需在该行后面加上“`//`”并标明注释部分即可。

(2)C 语言中,标识符是一个名字,一般采用小写字母作标识符。C 语言允许用作标识的字符有以下几种。

①26 个英文字母,包括大小写(共 52 个)。

②数字 0、1、……、9。

③下画线“_”。

④C 语言标识符由英文字母、数字和下画线组成,但不能以数字打头,标识符之间不能含有空格,长度为 1~32。

(3)C 语言区分大小写。如 SUM、sum 和 Sum 表示三个不同的标识符。

(4)C 程序书写格式灵活,一个语句可连续写在多行上,一行中也可以写多个语句。如例 1.2 中的“SUM=0;N=1;”两个语句可以写在一行上,也可以写在两行上。

(5)为了使书写的程序结构清晰、层次分明,建议采用“缩进对齐”的格式编辑 C 语言源程序。

1.2.4 C 程序的三种基本结构和流程图

C 语言程序有以下三种基本结构。

(1)顺序结构。如图 1.2 所示,虚线框内是一个顺序结构,其中 A 和 B 两个框是顺序执行的。顺序结构是最简单的基本结构。

(2)选择结构。如图 1.3 所示,虚线框内是一个选择结构。此结构包括一个判断框,根据给定的条件 P 是否成立而选择执行 A 框或 B 框。

(3)循环结构。循环结构是反复执行某一部分的操作。循环结构有以下两种。

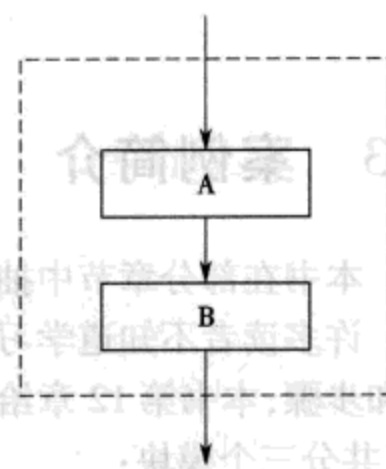


图 1.2 顺序结构

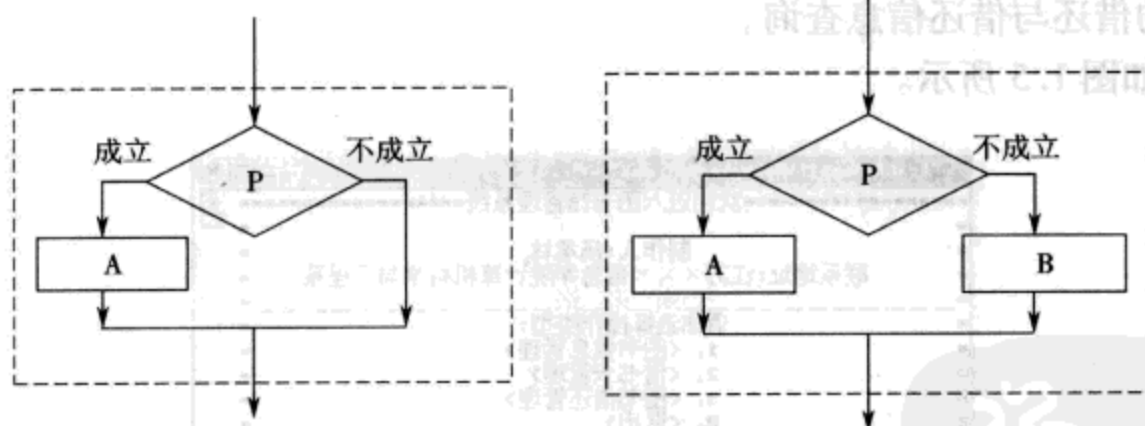


图 1.3 选择结构

①当型循环结构。如图 1.4(a),当条件 p1 成立时才执行 A 框操作,否则 A 框操作一次也不执行。

②直到型循环结构。如图 1.4(b),必须执行 A 框一次后,才去判定条件 p2 是否成立,成立继续执行 A 框操作,否则 A 框操作不执行。

三种基本结构具有以下共同特点:

(1)只有一个入口;

(2)只有一个出口;

(3)结构内的每一部分都有机会被执行;

(4)结构内不能存在“死循环”。

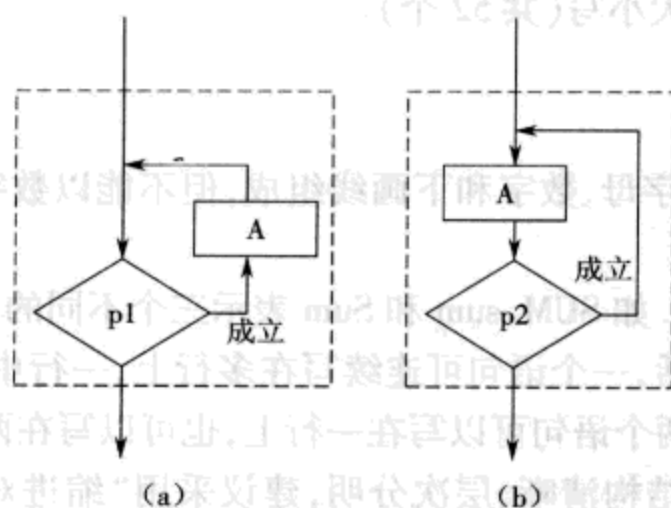


图 1.4 循环结构

(a) 当型循环结构; (b) 直到型循环结构

1.3 案例简介

本书在部分章节中抽取或改造了书后案例部分程序,以促进读者理解与掌握 C 语言的学习。许多读者不知道学习 C 语言到底应用在何处,更不知道如何编写一个完整的 C 程序的方法和步骤,本书第 12 章给出一个完整的“图书管理系统”的 C 语言程序源代码。“图书管理系统”共分三个模块:

- (1) 图书信息的增加、删除、修改和查询;
- (2) 借书卡信息的增加、删除、修改和查询;
- (3) 图书的借还与借还信息查询。

其主菜单如图 1.5 所示。

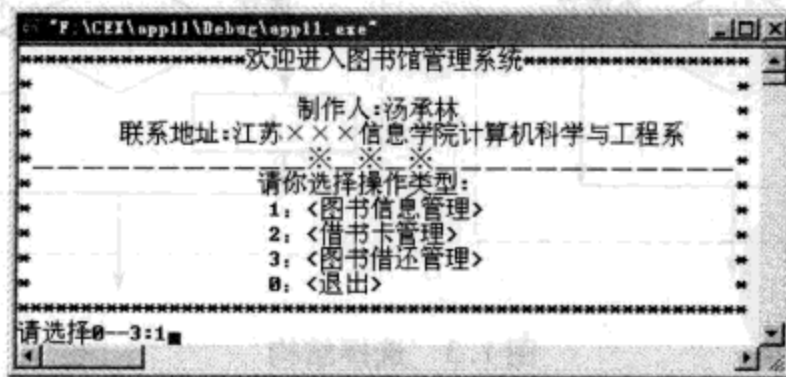


图 1.5 本书案例程序主菜单

1.4 Visual C++ 6.0 上机操作

1.4.1 C 程序编译与可执行文件的生成

C 语言是高级程序语言,用 C 语言写出的程序通常称为 C 源程序,人们容易使用、书写和阅读 C 源程序,但计算机却不能直接执行,因为计算机只能识别和执行特定的二进制形式机器语言程序。为使计算机能完成某个 C 源程序所描述的工作,就必须把 C 源程序转换成计算

机能够识别和执行的二进制机器语言。C 语言程序编译和执行过程如图 1.6 所示。

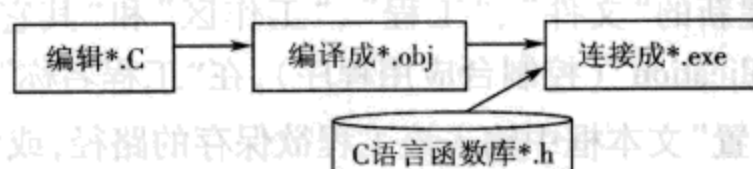


图 1.6 C 源程序转换成可执行文件

C 语言程序可执行文件生成过程如下。

(1) 利用编辑器编写生成 C 语言源程序(文本文件)。编辑器可用记事本、Word 等。

(2) 采用 C 编译器将源程序编译成二进制的目标文件(伪代码),生成的目标文件扩展名为 .obj。

(3) 采用 C 连接程序将目标文件(*.obj)与库文件(*.h)连接,生成可执行文件。可执行文件扩展名为 .exe,可执行文件可脱离 C 环境运行。

1.4.2 Visual C ++ 6.0 的上机操作步骤

全国计算机等级考试(二级 C)采用 Visual C ++ 6.0 环境。为了与全国计算机等级考试环境相匹配,本书的所有源程序皆用 Visual C ++ 6.0 编辑、编译与运行。

用 Visual C ++ 6.0 创建的 C 程序被存储为一个独立工程(Project),每个工程会新建一个文件夹,工程中含有一组文件,这组文件可以具有不同的扩展名,其中部分文件由 Visual C ++ 6.0 自动创建,这组文件组合在一起形成一个完整的应用程序。

下面以例 1.2 为例(例 1.2 的源程序文件名为 ex1_2.c)介绍 Visual C ++ 6.0 中建立工程并进行 C 程序调试的主要操作步骤。

1. 启动 Visual C ++ 6.0

如图 1.7 所示,从“开始”按钮中的“程序”项中启动“Microsoft Visual C ++ 6.0”。

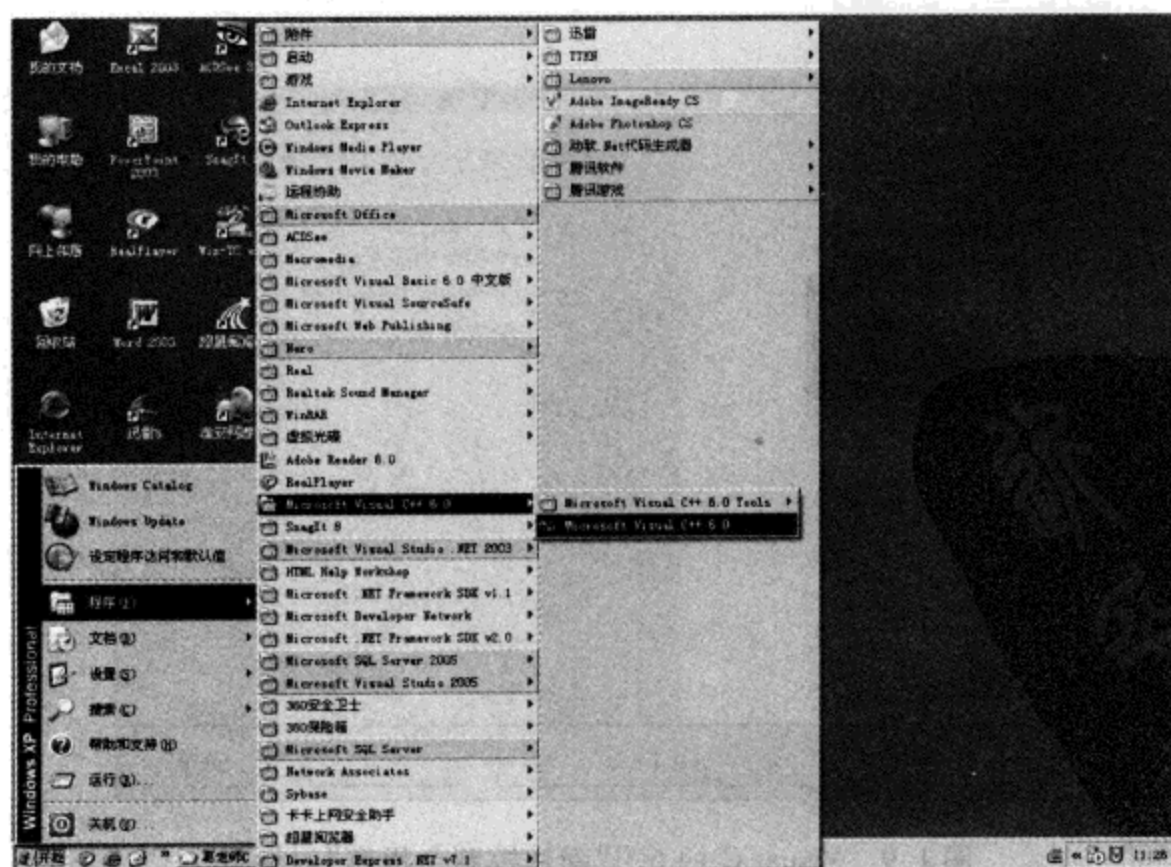



图 1.7 启动 Visual C ++ 6.0

2. 进入 Visual C ++ 6.0 环境

执行“文件”→“新建”菜单项,弹出“新建”对话框,如图 1.8 所示。该对话框有分别用于

创建新的“文件”、“工程”、“工作区”和“其它文档”等 4 个选项标签。选中“Win32 Console Application”(控制台应用程序),在“工程名称”文本框中输入欲建工程名称,如 ex1_2;然后在“位置”文本框中输入该工程欲保存的路径,或者单击其右边的  按钮,在弹出的“选择目录”对话框中选择保存路径。单击“确定”弹出如图 1.9 所示的界面,在图 1.9 中选择“一个空工程”后单击“完成”按钮。然后在“新建工程信息”(New Project Information)对话框中单击“确定”按钮即可。

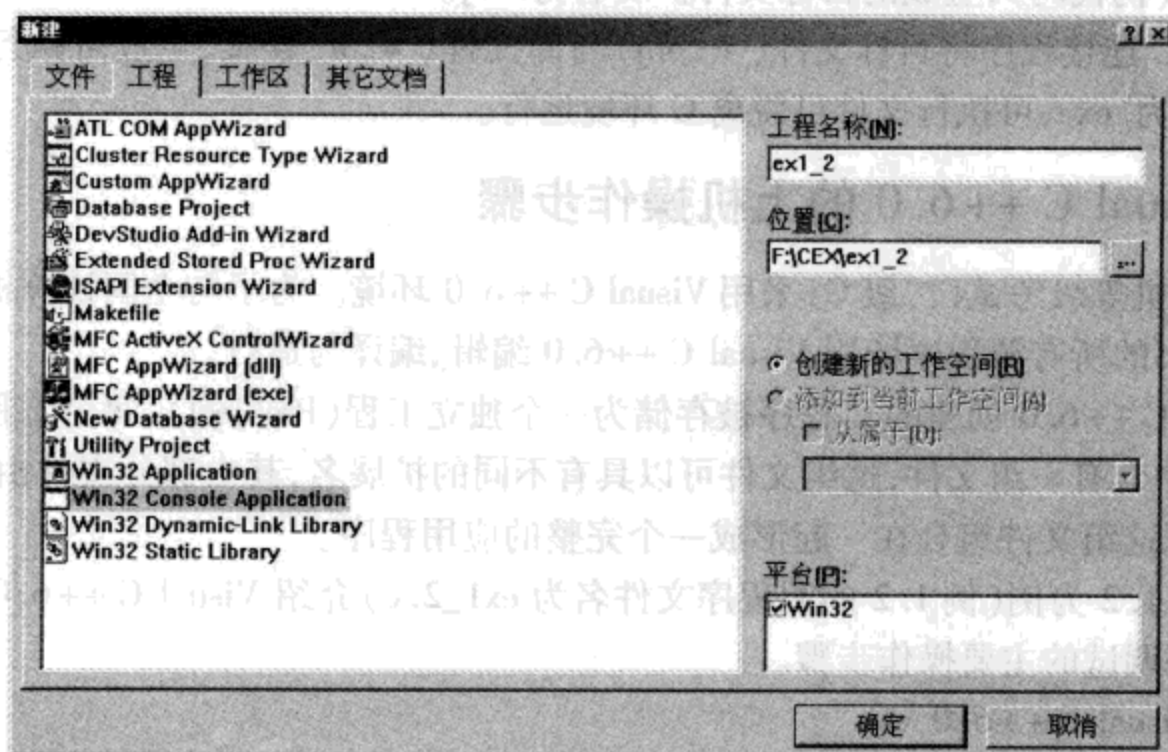


图 1.8 Visual C++6.0“新建”对话框

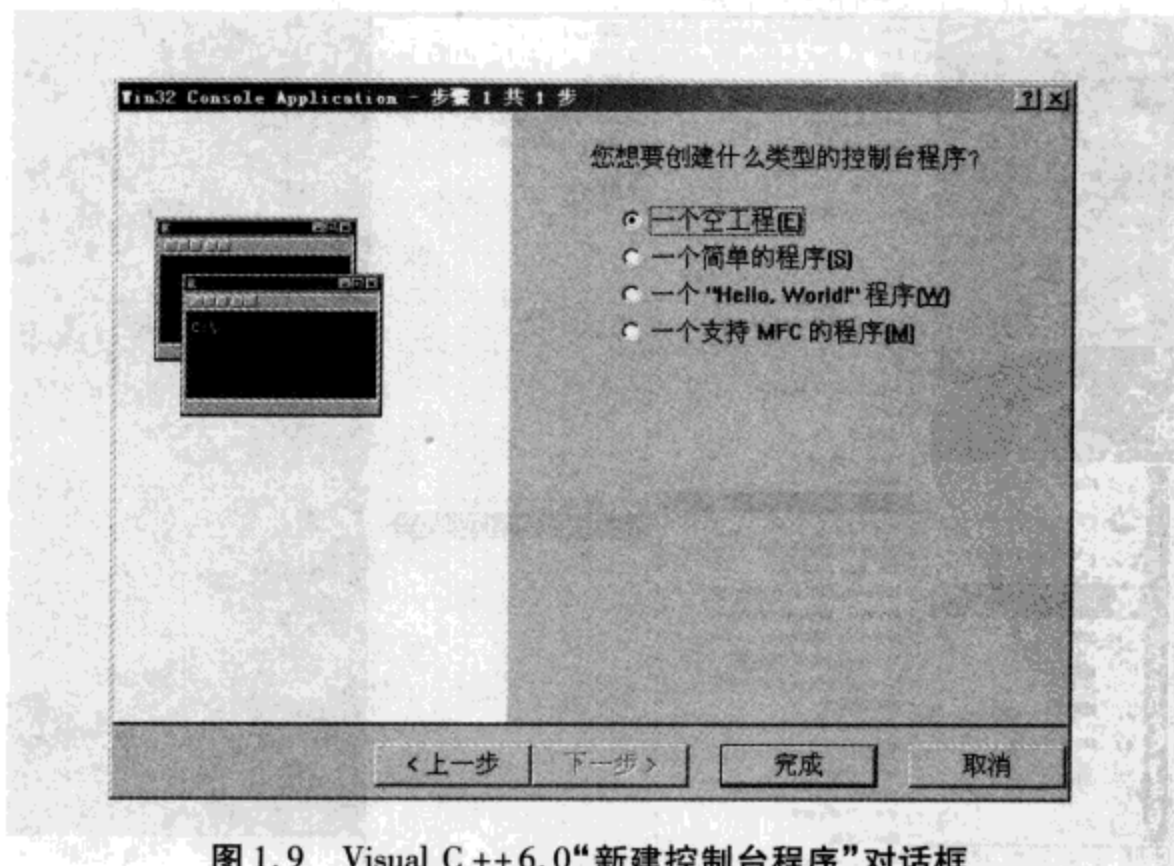


图 1.9 Visual C++6.0“新建控制台程序”对话框

3. 在工程中添加并编辑源程序

单击“新建”下拉菜单项,弹出的界面如图 1.10 所示,选择文件类型为 C++ Source File,输入源文件名(如 ex1_2.c,注意扩展名为.c,若不加则默认扩展名为.cpp),选择保存文件的

位置,单击“确定”按钮后将生成一个新的空文件 ex1_2.c,并弹出源文件编辑窗口,如图 1.11 所示。

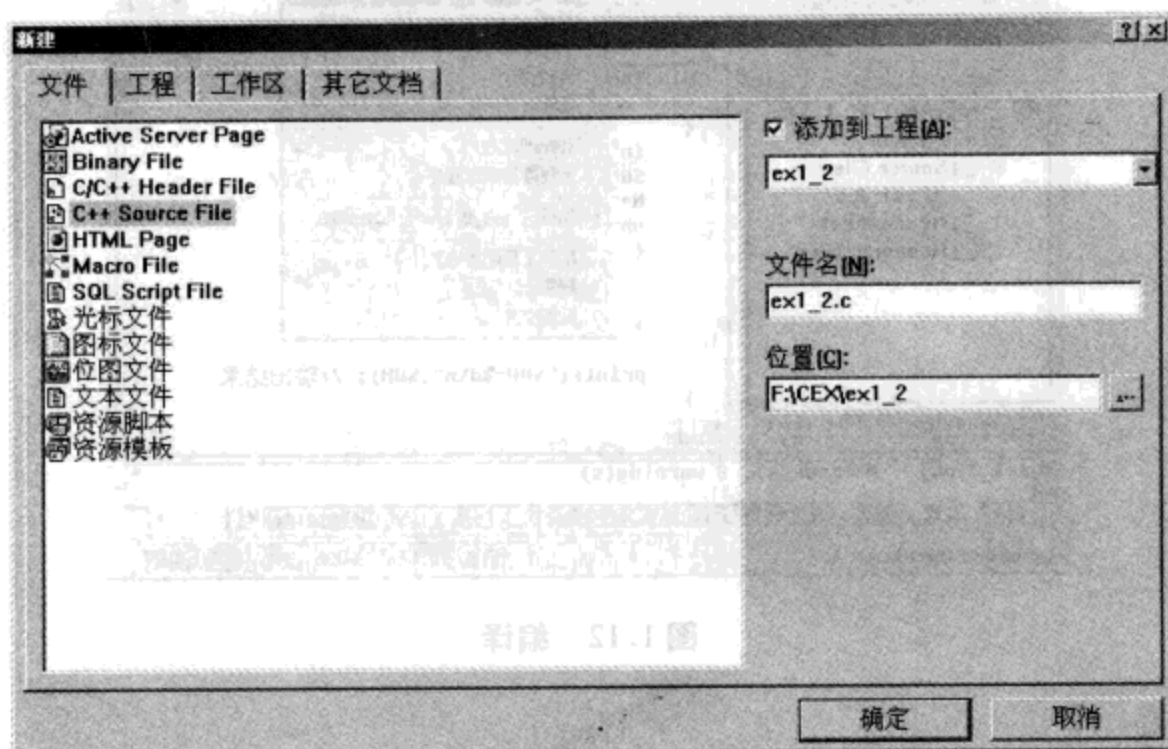


图 1.10 在工程中添加文件的对话框

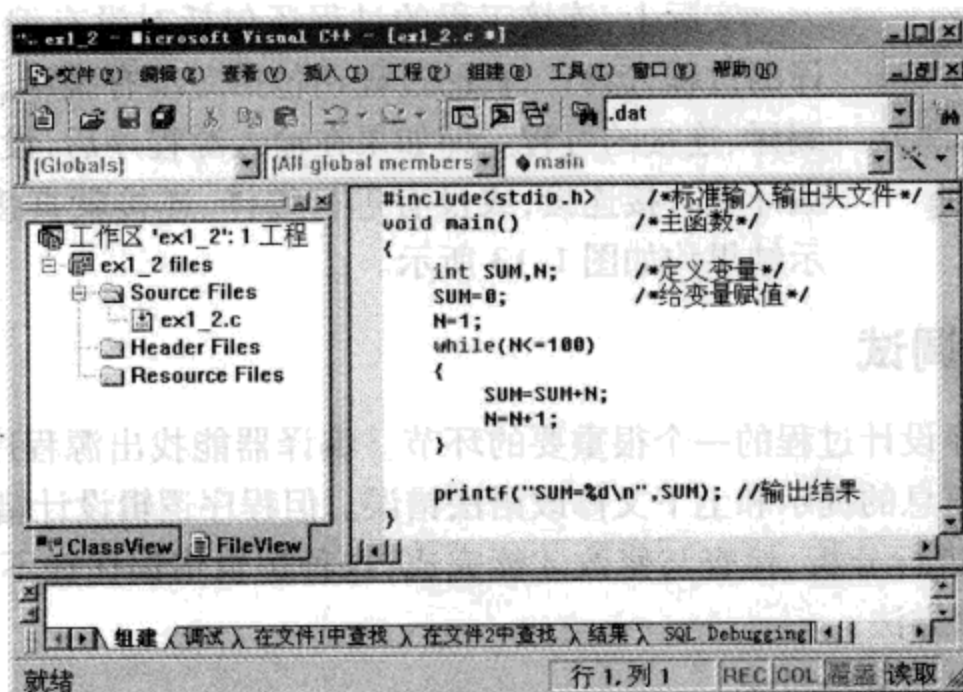




图 1.11 在编辑窗口编辑 C 源程序代码

4. 编译操作

如图 1.12 所示,选择“组建”菜单下的“编译”(comple)(或单击  按钮),对应的快捷方式为 Ctrl + F7,将生成 .obj 目标文件。编辑完成后在输出窗口显示“ex1_2.obj - 0 error(s), 0 warning(s)”,表示执行完后生成目标文件 ex1_2.obj,而且没有 error 错误,也没有 warning 错误。

5. 连接操作

选择“组建”菜单下的“组建”(Build)(或单击  按钮),对应的快捷方式为 F7,将生成 .exe 可执行文件。执行完成后显示“ex1_2.exe - 0 error(s), 0 warning(s)”,表示执行完后生成了可执行文件 ex1_2.exe,而且没有 error 错误,也没有 warning 错误。

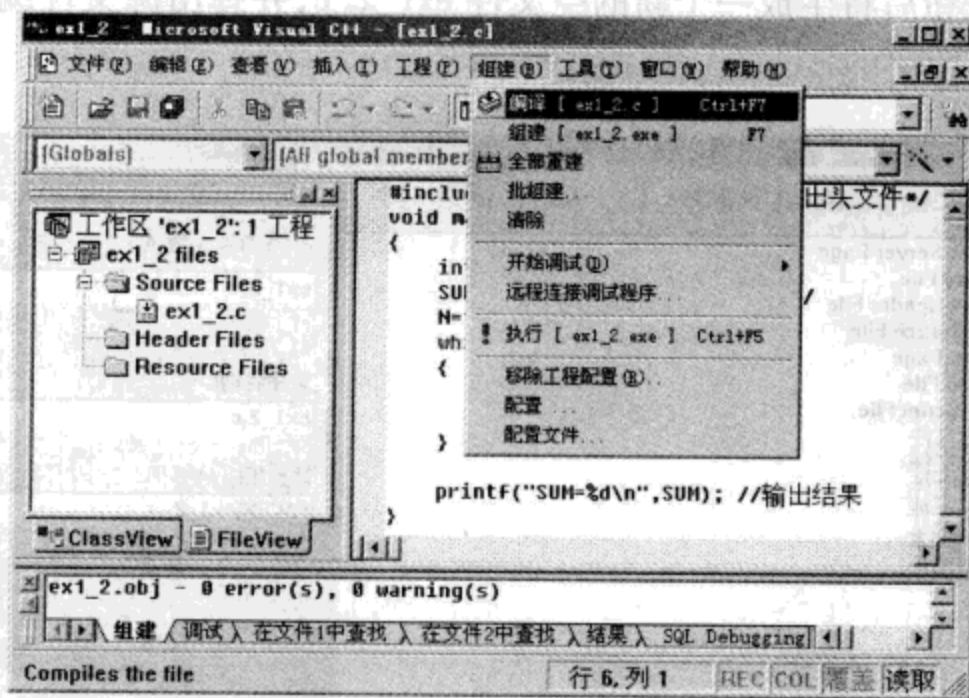



图 1.12 编译

6. 执行程序

选择“组建”菜单下的“执行”选项 (Build Execute) (或单击  按钮), 对应的快捷方式为 Ctrl + F5, 将运行 ex1_2.exe 文件。

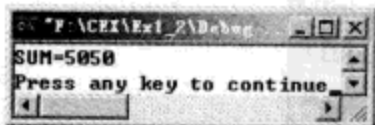


图 1.13 运行结果

实际上, 连接工程的过程还包括对没有编译的源文件进行编译的过程, 运行工程的过程还包括对没有编译、连接的源文件进行编译、连接的过程。即如果在输入源程序后, 没有对该源程序进行编译就直接连接, 或没有进行编译、连接就直接执行, 则输出的显示结果均如图 1.13 所示。

1.4.3 C 程序调试

程序调试是程序设计过程的一个很重要的环节。编译器能找出源程序中的语法错误, 程序员可以根据错误信息的提示和上下文修改语法错误。但程序逻辑设计错误只能靠程序员利用调试工具对程序进行分析、检查与修改才能完成, 这种逻辑错误往往不易发现。下面介绍 Visual C ++6.0 查错方法。

1. 查找源程序中的语法错误

C 语言程序的错误主要包括两大类: 一类是语法错误; 一类是逻辑设计错误。

语法错误是指违背了 C 语言语法规则而导致的错误。语法错误一般分为一般错误 (error) 和警告错误 (warning) 两种。

- (1) 当用户程序出现 error 错误时, 将不会产生可执行文件。
- (2) 当用户程序中出现 warning 错误时, 通常能够生成可执行文件, 但程序运行时可能发生错误, 严重的 warning 还会引起死机现象。

所以 warning 错误比 error 错误更难以修改, 应该尽量消除 warning 错误。

如果程序有语法错误, 则在编译时, Visual C ++6.0 的编译器将在输出窗口中给出语法错误提示信息, 如图 1.14 所示。错误提示信息一般还可以指出错误所在位置的行号。用户可以在输出窗口中双击错误提示信息或按 F4 键返回到源程序的编辑窗口, 并通过一个定位查到引

起错误的语句。

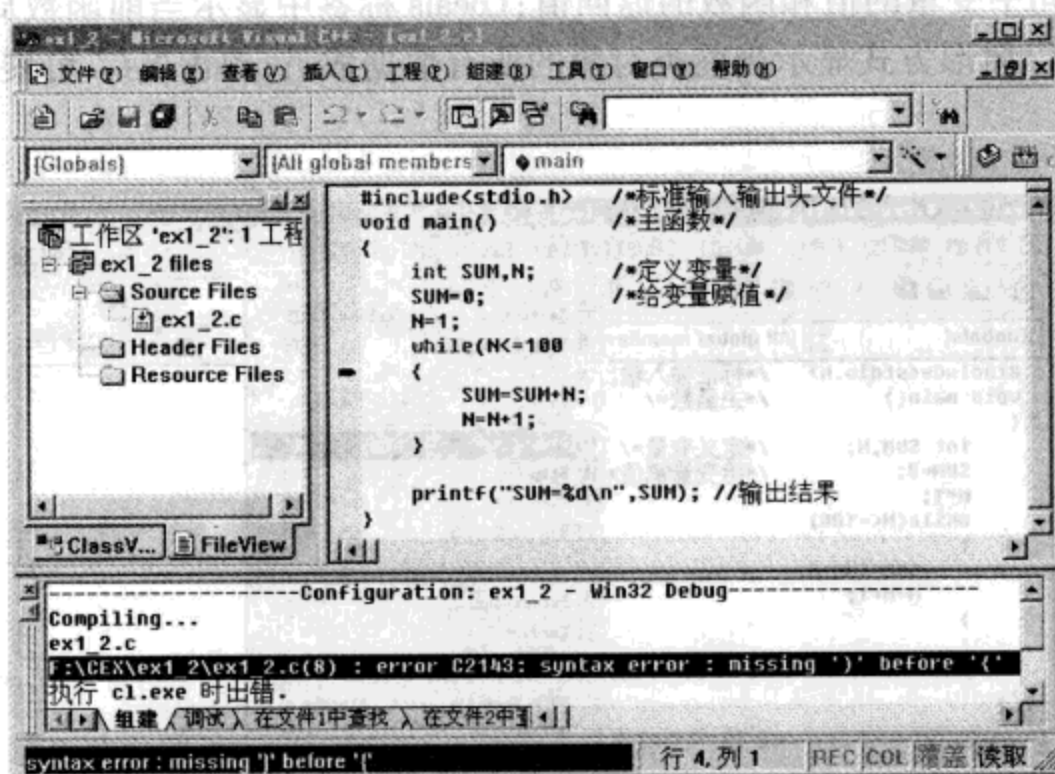


图 1.14 C++ 调试源程序时出错提示

错误提示信息位置不十分准确,并且一处错误往往会引出若干条错误提示信息。因此,修改一个错误后最好马上进行程序的编译或运行,例如在图 1.14 中,错误提示信息中括号内的数字 8 指示错误发生在第 8 行,指出了错误的箭头也指向第 8 行,但实际错误发生在第 7 行末尾,因为第 7 行末尾少一个“)”圆括号。

如果程序并没有违背 C 语言语法规则,编译器也没有提示出错,而且程序能够成功运行,但程序执行结果与原意不符,这类程序设计上的错误被称为逻辑设计错误或缺陷(Bug),这类错误由于编译器不能给出错误提示,所以必须利用“调试器”(Debug)对程序进行跟踪调试才能发现错误。

2. 调试器(Debug)

Visual C++ 6.0 提供了一个重要的工具——调试器,用于查找和修改程序中逻辑设计错误。在“组建”菜单下选择“开始调试”子菜单下的 Go、Step Into、Run To Cursor、Step Over 及附加到当前进程 5 个菜单项,它们的功能如表 1.3 所示。

表 1.3 开始调试(D)子菜单中的菜单项和功能

菜单项	快捷键	功 能
Go	F5	程序运行到某个断点、程序的结束或需用户输入的地方
Run to Cursor	Ctrl + F10	程序执行到当前光标处
Step into	F11	单步执行程序的一条指令,能进入被调用的函数内部
Step Over	F10	单步执行程序的一条指令,不进入被调用的函数内部
附加到当前进程(A)		将调试器与一个正在运行的进程相连接

一旦调试过程开始后,“调试”(Debug)主菜单将取代“组建”主菜单项出现在主菜单中,

同时出现一个可停靠的调试工具栏和一些调试窗口,如图 1.15 所示。Auto 标签中显示当前语句或前一条语句中变量的值和函数的返回值;Locals 标签中显示当前函数局部变量的名称、值和类;this 标签以树形方式显示当前类对象的所有数据成员,单击“+”可展开 this 指针所指对象。

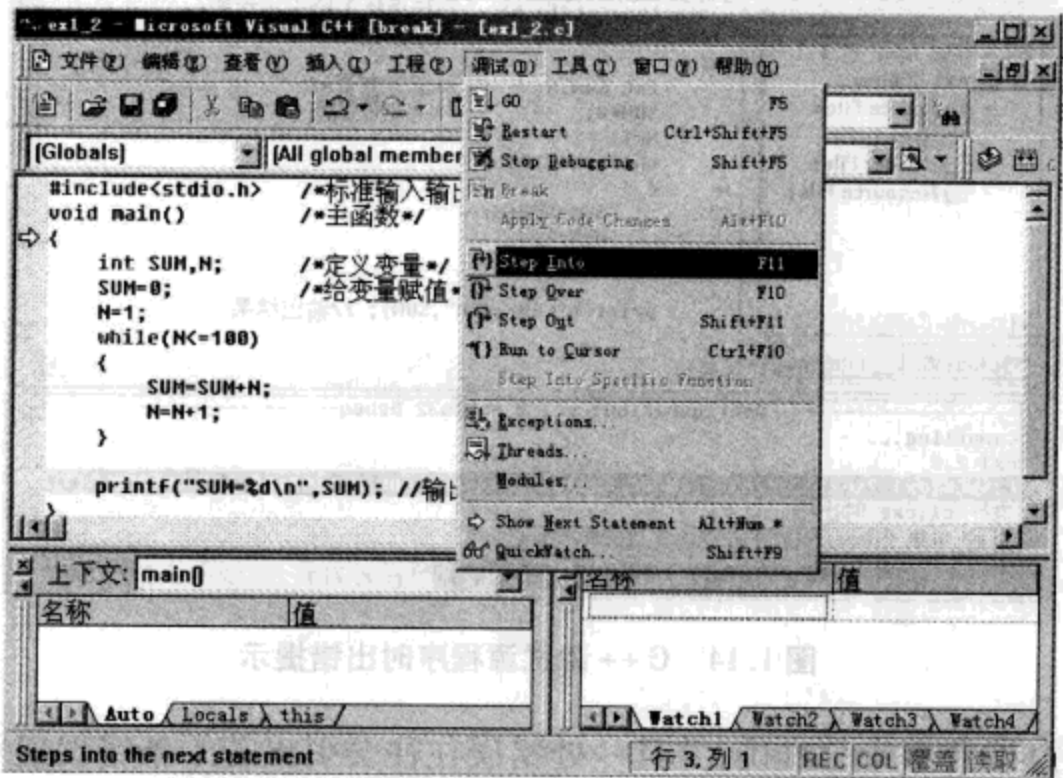


图 1.15 C++ 调试程序界面 1

Watch 窗口用于观察和修改变量或表达式的值,它拥有 Watch1、Watch2、Watch3、Watch4 等 4 个标签。在每个标签中,用户都必须手工设置要观察的变量或表达式。如图 1.16 中 Watch1 中设置的观察变量 SUM 和 N,其对应的值为 1 和 1,显示的是执行第一次循环的变量的结果。

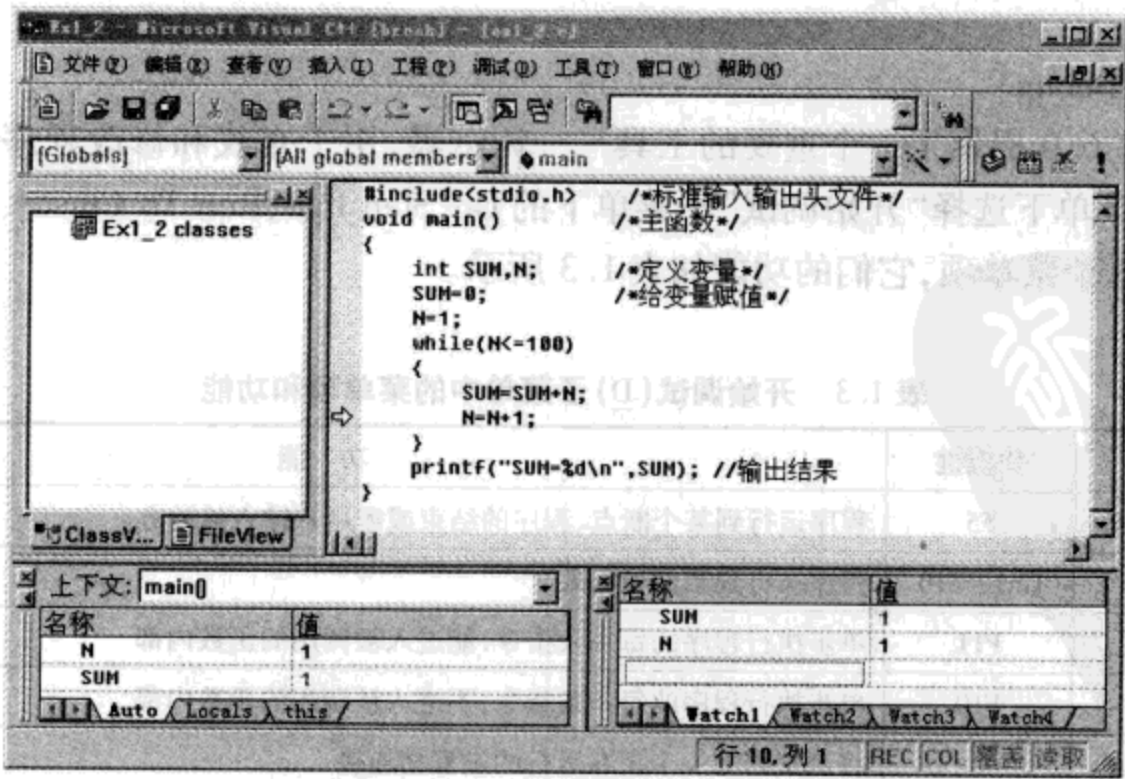


图 1.16 C++ 调试程序界面 2

3. 跟踪调试程序

通常程序跟踪调试的基本原理就是在程序运行过程中设计断点,观察某一阶段变量的状态。因此,跟踪调试要做的第一件事就是要使程序在某一点(运行到某条语句时)停下来。用户首先要设置断点,再运行程序,当程序在断点设置处停下来时,再利用各种工具观察程序在此时的状态。

1) 设置断点

设置断点的最简单方式是将鼠标移到目标位置后点右键,然后在右键菜单中单击“Insert/Remove”即可在该行设置断点。另外,可以选择“编辑(E)”菜单下的“断点……”菜单项,系统将显示“Breakpoints”对话框。

2) 控制程序运行

当设置断点后,程序就可以进入调试状态,并按要求控制程序的运行,其中有4条命令:Step Over、Step Into、Step Out、Run to Cursor。这4条命令的功能与调试菜单中相应菜单项功能一致,用户可以通过用鼠标单击工具栏按钮或使用热键来控制程序的运行。

3) 观察数据变化

在调试过程中,用户可以通过 Watch 窗口和变量窗口查看当前变量的值,这些信息可以反映程序运行过程中的状态变化以及变化结果的正确与否,可以反映程序是否有错,再加上人工分析,就可以发现错误所在。

Visual C++6.0 是功能强大的可视化 C/C++ 集成开发环境,用户只有通过多练习方能熟练、灵活和全面掌握其功能,一旦用户掌握了 Visual C++6.0 的应用方法和技巧,就能快速、高质量地完成各种 C/C++ 语言程序的设计工作。

本章小结

(1) 程序是指按照计算机程序设计语言规范书写出来的一系列有序且有限的语句,它表达了编程人员要求计算机执行的操作。

(2) 算法是指解决方案及对方案准确、完整的描述。

程序 = 数据结构 + 算法

(3) C 语言有如下特点:过程化、模块化、结构化、灵活性、可移植性、丰富的数据类型和运算符等。

(4) C 语言的生成与运行机制包括编辑、编译、连接和运行4个步骤。

习题一

一、选择题

1. 以下不能定义为用户标识符的是()。
A) Main B) _0 C) _int D) sizeof
2. 以下几组选项中,均为不合法的标识符是()。
A) A, P_0, do B) float, la0, _A C) b-a, goto, int D) _123, temp, INT
3. 下列可用于 C 语言用户标识符的一组是()。

A) void, define, WORD

B) a3_b3, _123, Car

C) For, -abc, IF Case

D) 2a, DO, sizeof

4. 下列叙述中错误的是()。

A) 计算机不能直接执行用 C 语言编写的源程序

B) C 程序经 C 编译程序编译后,生成后缀为 .obj 的文件是一个二进制文件

C) 后缀为 .obj 的文件,经连接程序生成后缀为 .exe 的文件是一个二进制文件

D) 后缀为 .obj 和 .exe 的二进制文件都可以直接运行

5. 对于一个正常运行的 C 程序,以下叙述中正确的是()。

A) 程序的执行总是从 main() 函数开始,在 main() 函数结束

B) 程序的执行总是从程序的第一个函数开始,在 main() 函数结束

C) 程序的执行总是从 main() 函数开始,在程序的最后一个函数中结束

D) 程序的执行总是从程序的第一个函数开始,在程序的最后一个函数中结束

6. C 语言源程序名的后缀是()。

A) .exe

B) .c

C) .obj

D) .cp

7. 以下叙述中正确的是()。

A) C 程序中的注释只能出现在程序的开始位置和语句的后面

B) C 程序书写格式严格,要求一行内只能写一个语句

C) C 程序书写格式自由,一个语句可以写在多行上

D) 用 C 语言编写的程序只能放在一个程序文件中

8. 以下叙述中正确的是()。

A) C 程序的基本组成单位是语句

B) C 程序中的每一行只能写一条语句

C) 简单 C 语句必须以分号结束

D) C 语句必须在一行内写完

9. 计算机能直接执行的程序是()。

A) 源程序

B) 目标程序

C) 汇编程序

D) 可执行程序

10. 下列说法正确的是()。

A) 在书写 C 语言源程序时,每个语句以逗号结束

B) 注释时,“/”和“*”号间可以有空格

C) 无论注释内容多少,在对程序编译时都被忽略

D) C 程序每行只能写一个语句

11. 一个完整的可运行的 C 源程序中()。

A) 可以有一个或多个主函数

B) 必须有且仅有一个主函数

C) 可以没有主函数

D) 必须有主函数和其他函数

12. 下列叙述正确的是()。

A) 在执行 C 程序时,不是从 main() 函数开始

B) C 程序书写格式限制严格,一行内必须写一个语句

C) C 程序书写格式比较自由,一个语句可以分在多行上写

D) C 程序书写格式严格,要求一行内必须写一个语句,并要有行号

13. 以下叙述中正确的是()。

A) 在 C 程序中无论整数还是实数,只要在允许的范围内都能准确无误地表示

B) C 程序由主函数组成

C) C 程序由函数组成

D) C 程序由函数和过程组成

二、填空题

1. C 语言规定,标识符只能由____、____、____三种字符组成,而且第一个字符必须是____或____。

2. 一个 C 程序一般由若干个函数构成,程序中至少包括一个_____。

3. 一个 C 程序总是从_____开始执行。

4. C 程序编译后生成_____程序,连接后生成_____程序。

三、编程题

熟悉 Visual C++ 6.0 环境,在屏幕上输出“这是我编写的第一个 C 程序!”。

2.1 基本数据类型

在 C 语言中,数据类型是基本数据类型,它决定了数据的表示方法和运算规则。C 语言的基本数据类型包括整型、浮点型、字符型和枚举型。整型分为有符号整型 (signed) 和无符号整型 (unsigned), 浮点型分为单精度浮点型 (float) 和双精度浮点型 (double), 字符型为 char, 枚举型为 enum。在 Visual C++ 6.0 中, 整型的大小取决于编译选项, 默认情况下, 有符号整型为 16 位, 无符号整型为 16 位, 单精度浮点型为 32 位, 双精度浮点型为 64 位, 字符型为 8 位, 枚举型为 16 位。在 Turbo C 2.0 中, 有符号整型为 16 位, 无符号整型为 16 位, 单精度浮点型为 32 位, 双精度浮点型为 64 位, 字符型为 8 位, 枚举型为 16 位。在 Visual C++ 6.0 中, 有符号整型为 16 位, 无符号整型为 16 位, 单精度浮点型为 32 位, 双精度浮点型为 64 位, 字符型为 8 位, 枚举型为 16 位。

表 2.1 基本数据类型

数据类型	关键字	取值范围	存储长度 (字节)
有符号整型	int	-32768 ~ 32767	2
无符号整型	unsigned int	0 ~ 65535	2
有符号短整型	short	-32768 ~ 32767	2
无符号短整型	unsigned short	0 ~ 65535	2
有符号长整型	long	-2147483648 ~ 2147483647	4
无符号长整型	unsigned long	0 ~ 4294967295	4
单精度浮点型	float	3.4E-38 ~ 3.4E+38	4
双精度浮点型	double	1.7E-308 ~ 1.7E+308	8
字符型	char	-128 ~ 127	1
枚举型	enum	由用户定义	2

第 2 章 数据类型、运算符及表达式

在第 1 章中提到了数据结构和算法的概念,数据结构是数据的组织形式,而数据是算法操作的对象,操作的目的是对数据的加工处理,以得到期望的结果。数据是以某种特定的形式(如整数、实数、字符等)存在的,不同的数据之间还存在某种联系(例如由若干实数组成的一个实数数组)。不同形式的数据占用不同的存储空间,C 语言的数据结构是以数据类型的形式体现的。C 语言定义了 4 种基本数据类型:字符型(char)、整型(int)、单精度浮点型(float)和双精度型(double)。它们是构造其他数据类型的基础,其值不可以再分解为其他类型。

2.1 基本数据类型

计算机中,所有的数据都采用二进制形式表示,4 种基本数据类型规定了数据在内存中占用的二进制位数(或字节数)。与数据类型相关的是类型修饰符,对于整数类型有两类修饰符,一类是符号修饰符,一类是长度修饰符。其中符号修饰符有带符号修饰符(signed)和不带符号修饰符(unsigned)之分,默认为带符号修饰符;长度修饰符有短型(short)和长型(long)之分,数据长度与具体的机器编译环境有关。如 Visual C++ 6.0 编程环境下,int 型与 long 型一样,都是占用 4 字节(32 bit);而在 Turbo 2.0 编程环境下,int 型与 short 型一样,都是占 2 字节(16 bit)。本书以 Visual C++ 6.0 作为开发环境,其主要数据类型和取值范围如表 2.1 所示。表中类型带方括号的项表示是格式中的可选项。

表 2.1 基本数据类型及取值范围

类型		取值范围	长度(字节)	有效数字(位)
整数	[signed] short [int]	$-2^{15} \sim 2^{15} - 1$	2	
	unsigned short [int]	$0 \sim 2^{16} - 1$		
	[signed] int	$-2^{31} \sim 2^{31} - 1$	4	
	unsigned int	$0 \sim 2^{32} - 1$		
	[signed] long [int]	$-2^{31} \sim 2^{31} - 1$		
	unsigned long [int]	$0 \sim 2^{32} - 1$		
单精度浮点数	float	$10^{-37} \sim 10^{38}$	4	6 ~ 7
双精度浮点数	double	$10^{-307} \sim 10^{308}$	8	15 ~ 16
字符型	[signed] char	$-2^7 \sim 2^7 - 1$	1	
	unsigned char	$0 \sim 2^8 - 1$		

【知识链接】

2.2 常量与变量

2.2.1 常量与变量定义

C 语言中数据分为两类:常量与变量。

1. 常量的定义

常量是指在程序运行过程中数值不变的量。常量通常是数值型(整数及带小数的实数),也可以是字符或字符串,例如:

整型常量 1、101、-21、6 等;

实型常量 -123.45、-1.0、34.56 等;

字符常量 'A'、'b'、'@' 等;

字符串常量 "computer"、"这是我编写的 C 语言程序"等。

2. 变量的定义

变量是指程序运行过程中,可以发生变化的量。编写程序时,需要将数据存储在内存中,以方便使用或修改这个数据的值。在程序中,通常使用变量来存储数据。

在 C 语言中,变量的使用必须首先经过定义。变量的定义形式如下:

[存储类型] 数据类型 变量名 1[, 变量名 2, ...];

其中方括号中的内容为可选项,可以同时定义多个相同类型的变量,之间用逗号分开。存储类型将在后续章节中讲述。变量具有 3 个要素:变量名称、类型和值。

例如:

```
int a, b, c, _abc, a5;
```

```
float _b1, abc, A2;
```

其中,变量的数据类型可以是表 2.1 中的任何数据类型,变量命名必须遵守 C 语言标识符的命名规则:

(1) 第 1 个字符必须是非数字字符;

(2) 其余字符可以是字母、下画线和数字;

(3) 字母区分大小写;

(4) 用户自定义标识符与 C 语言保留字或预定义标识符不同名,并应尽可能做到“见名知意”,以增强程序的可读性。

根据上述命名规则, name、age、student_id、_xyz 都是合法的变量名。

3. 初始化变量

变量的初始化是指在定义变量的同时给变量赋一个初值。初始化定义的形式如下:

[存储类型] 数据类型 变量名 1 = 初值 1[, 变量名 2 = 初值 2, ...];

例如:

```
int a = 1, b = 2, c = 3, _abc = 0, a5 = 1;
```

```
float _b1 = 1.2, abc = -1.2;
```

```
char ch0 = 'A', ch1 = 'h';
```

【例 2.1】 编写一个 C 语言程序,假如圆的半径为 3,计算圆的面积。

【解题思路】

已知圆的半径 r , 求圆的面积 s , 则求圆的面积公式为 $s = \pi r^2$, 其中 π 是常量, $\pi = 3.14$, 而 $r = 3$, 则圆的面积 $s = 3.14 \times 3^2 = 28.26$, 在此可定义一个变量 r 用来保存半径的值, 一个变量 s 保存圆的面积值。变量必须先定义后使用, 即要先为它们开辟内存单元。

【参考代码】

```
#include <stdio.h>
void main()
{
    float r,s;
    r=3;
    s=3.14*r*r;
    printf("s=%f\n",s);
}
```

```
/* 定义半径和面积变量 */
/* 给半径赋值 */
/* 计算圆面积 */
//输出面积值
```

程序运行结果如图 2.1 所示。

【注意点】

(1) “float r,s;”语句是用来声明 r 和 s 变量的, 其中 float 是 C 语言中的关键字或保留字, 表示 r 和 s 是单精度浮点实数, 这在后面实数类型中介绍。

(2) “s=3.14*r*r;”语句中 3.14 是常量, “*”是乘法运算符, “=”是把 $3.14 * r * r$ 的结果存放在左边的变量 s 所在的内存单元中, 不能理解为中小学数学中的“ s 等于 $3.14 * r * r$ ”。例如: “s=1; s=s*2;”语句中 $s=s*2$, 在中小学知识中就无法理解, 但在 C 语言中这种表达是正确的, 应理解为 s 的值乘以 2 所得的结果保存在 s 所在的内存单元中或称为赋给 s 。

(3) “printf(“s=%f\n”,s);”语句的作用是输出 s 的值, 其中“s=%f”表示一种输出格式, “%f”的作用是指明与之对应的变量 s 的输出格式, “f”表示为单精度浮点数。注意“%%”仅表示输出一个“%”。

(4) 常量的定义还可以使用符号常量的概念。在 C 语言中, 可以用一个标识符来表示一个常量, 称之为符号常量。符号常量在使用前也必须先定义, 其一般形式为

```
#define <符号常量名> <常量值>
```

其中, #define 是一条预处理命令, 也称宏定义命令, 其功能是把该标识符定义为其后的常量值。一经定义, 以后在程序中所有出现该标识符的地方均代之以该常量值。习惯上符号常量的标识符用大写字母表示, 变量标识符用小写字母表示, 以示区别。本例程序可改写如下:

```
#include <stdio.h>
```

```
#define PI 3.14
```

```
void main()
```

```
{
```

```
    float r,s;
```

```
    r=3;
```

```
    s=PI*r*r;
```

```
    printf("s=%f\n",s);
```

```
}
```

```
/* 定义符号常量 */
```

```
/* 定义半径和面积变量 */
```

```
/* 给半径赋值 */
```

```
/* 计算面积 */
```

```
//输出面积值
```

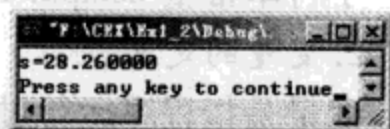


图 2.1 例 2.1 结果

(1) 十进制整数。如 321, -654, 0。

(2) 八进制整数。以 0(零)开头的数是八进制数。如 0321 表示八进制数 321,也可表示为 $(321)_8$, 其值为 $3 \times 8^2 + 2 \times 8^1 + 1 \times 8^0$, 等于十进制的 116。-012 表示八进制数 -12, 也可表示为 $-(12)_8$ 。

(3) 十六进制整数。以 0x(或 0X) 开头(0 为零)的数是十六进制数。如 0x12, 代表 16 进制数 12, 即 $(12)_{16} = 1 \times 16^1 + 2 \times 16^0 = 18$ 。-0x123 等于十进制数 -291。

整型常量按长度分为短整型常量和长整型常量(后缀为小写字母l或大写字母L),例如:

234、0x234、0234 分别是十进制、十六进制、八进制的有符号整数；

234L、0x234l、0234L 分别是十进制、十六进制、八进制的有符号长整数。

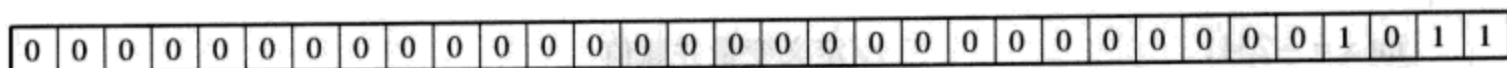
2. 整型变量

定义一个整型变量 i:

```
// 定义为整型变量
```

```
//给 i 赋以整数 100
```

十进制数 11 的二进制形式为 1011, 在 Visual C++ 6.0 中, 每一个整型数占有内存 4 个字节 (32 bit)。



实际上,数值是以补码表示的。一个正数的补码等于其本身。求负数的补码的方法是:将该数的绝对值的二进制形式,按位取反,末位加1。例如求-11的补码:①取-11的绝对值11;②11的二进制形式为1011;③对1011取反得111111111111111111111111111111110100(一个整数占32位);④再在末位加1得111111111111111111111111111111110101,如图2.3所示。



补码知识不属于本书讲解范围,但学习 C 语言或其他高级语言者必须了解数据在内存中的表示形式。

例如：

```
int a,b,c;           //a,b,c 为整型变量
long x,y;            //x,y 为长整型变量
unsigned short p,q;   //p,q 为无符号短整型变量
```

3. 整型数的输入/输出

整型数的输入/输出在输入函数 scanf() 及输出函数 printf() 中使用。整型数的输入/输出规则如表 2.2 所示。

表 2.2 整型数在 scanf() 和 printf() 函数中常用的转换字符串列表

转换字符串	规则说明
%d,%ld	分别表示输入/输出一个十进制整型数和长整型数
%x,%lx	分别表示输入/输出一个十六进制整型数和长整型数,x 可大写,则输出的十六进制输出符号也为大写,用这种方法可以清楚地看到一个整型数在内存中的二进制补码以十六进制的形式显示的结果
%u,%lu	分别表示输入/输出一个十进制无符号整型数和长整型数
%o,%lo	分别表示输入/输出一个八进制整型数和长整型数
%wd,%wld	分别表示输入/输出一个给定宽度 w 的十进制整型数和长整型数,如果 w>0 表示右对齐,如果 w<0 表示左对齐。(十六进制用 %wx,%wlx,八进制用 %wo,%wlo,无符号用 %wu,%wlu)。w 比实际值小的情况,按值的大小原样输出,w 不起作用

【例 2.2】 把整型数 -234 按表 2.2 格式输出。

【参考代码】

```
#include <stdio.h>
void main()
{
    int x = -234;           //定义变量并赋值
    printf("x = %d\n",x);   //十进制输出
    printf("x = %10d\n",x);
    printf("x = % -10d\n",x);
    printf("x = %2d\n",x);
    printf("x = % -2d\n",x);
    printf("x = %X\n",x);   //十六进制输出
    printf("x = %10x\n",x);
    printf("x = % -10x\n",x);
    printf("x = %2X\n",x);
    printf("x = % -2x\n",x);
    printf("x = %o\n",x);   //八进制输出
    printf("x = %10o\n",x);
    printf("x = % -10o\n",x);
    printf("x = %2o\n",x);
    printf("x = % -2o\n",x);
    printf("x = %u\n",x);   //无符号整型数输出
    printf("x = %10u\n",x);
```

```
printf("x = % -10u| \n",x);
printf("x = %2u| \n",x);
printf("x = % -2u| \n",x);
}
```

程序运行结果如图 2.4 所示。

【注意点】

(1) `printf("x = %X| \n",x)` 语句的“X”表示十六进制整型数输出是大写字母。

(2) `printf("x = %2d| \n",x)` 语句的输出宽度仅为 2,按正确数据宽度输出。

(3) `printf("x = %X| \n",x)` 中“|”仅用于查看结果对齐用,别无他意。

(4) `printf("x = %X| \n",x)` 中“\n”表示输出一行后换行,“x=”是普通字符,原样输出。

【例 2.3】 显示各种整数类型在内存中所占的字节数。

【解题思路】

一个整型数变量在内存中占多少字节,可用 `sizeof` 运算符计算,其格式为:

`sizeof(数据类型);`

或

`sizeof(变量名);`

【参考代码】

```
#include <stdio.h>
void main()
{
    short a;
    int b;
    long c;
    unsigned d;
    printf("短整型数宽度:%d\n",sizeof(short));
    printf("整型数宽度:%d\n",sizeof(int));
    printf("长整型数宽度:%d\n",sizeof(long));
    printf("无符号整型数宽度:%d\n",sizeof(unsigned));
    printf("短整型变量 a 宽度:%d\n",sizeof(a));
    printf("整型变量 b 宽度:%d\n",sizeof(b));
    printf("长整型变量 c 宽度:%d\n",sizeof(c));
    printf("无符号整型变量 d 宽度:%d\n",sizeof(d));
}
```

程序运行结果如图 2.5 所示。

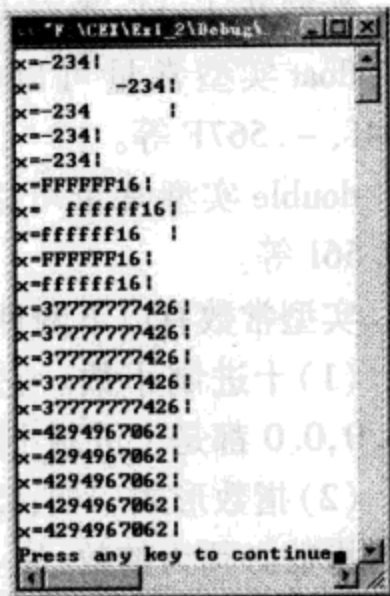


图 2.4 例 2.2 运行结果

2.2.3 实型数据

1. 实型常量

在 Visual C ++6.0 中用 4 字节或 8 字节来存储实数,分别用 float 类型及 double 类型表示,见表 2.1 所示。

float 实型常量可以在数的后面加上 f 或 F 来表示,例如 3.14f、-.567F 等。

double 实型常量可以在数的后面加上 l 或 L,例如 3.14L、-5.56l 等。

实型常数有以下两种表示形式。

(1)十进制小数。它由数字和小数点组成(注意必须有小数点)。如:.456,-0.456,456.0,0.0 都是十进制小数形式。

(2)指数形式。如:456e2,456E + 2 都表示 456×10^2 。注意 e(或 E)之前必须有数字,之后的指数必须为整数,如 e2,12.4e3.5,.5e,.e2,E 都是不合法的指数形式。

一个实数有多种表示形式。如 12.34 可以表示为 12.34e0,123.4e - 1,1.234e1,0.1234e + 2 等等。其中 1.234e1 称为“规范化指数形式”,小数点前仅有一位非零数字。一个实数在用指数形式输出时,是按规则化的指数形式输出的。

2. 实型变量

实型变量用 float 及 double 来定义。其一般形式为:

```
float <变量名>;
double <变量名>;
```

例如:

```
float a,b,c;           // a,b,c 为单精度实型变量
double xy,yz,zx;       // xy,yz,zx 为双精度实型变量
```

在定义变量的同时可以对它们赋初值。例如:

```
float a = 1.2,b = 2.0,c = -1.0/2; // a,b,c 为单精度实型变量
double xy = 1.1,yz = 0.0,zx = -5.0; // xy,yz,zx 为双精度实型变量
```

3. 实型数的输入/输出

实型数的输入/输出在输入函数 scanf() 及输出函数 printf() 中使用。实型数的输入/输出规则如表 2.3 所示。

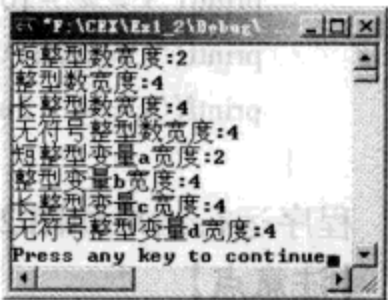


图 2.5 例 2.3 运行结果

表 2.3 实型数在 scanf() 和 printf() 函数中常用的转换字符串列表

转换字符串	规则说明
%f,%lf	分别表示输入/输出一个 float 型实数和 double 型实数
%m.nf,%m.nlf	分别输出一个十进制 float 型实数和 double 型实数,输出总宽度为 m,小数位数 n(n ≥ 0)。如果 m > 0 则右对齐,如果 m < 0 则左对齐;如果 m 的宽度小于实际实型数占的宽度,则按实际宽度输出,小数位数一定时,按四舍五入原则进行,如 n = 0 则表示不输出小数位。注意输出的符号、小数点都要各占一位

续表

转换字符串	规则说明
%m.ne,%m.nle	分别按指数格式输出一个float型实数和double型实数,其中指数部分占5位(例e+001),e占1位,指数符号占1位,指数占3位,小数占n位,总宽度占m位。如果m>0则右对齐,如果m<0则左对齐,如果m的宽度小于实际实型数占的宽度,则按实际宽度输出,小数位数一定时,按四舍五入原则进行,如n=0则表示不输出小数位。注意输出的符号、小数点都要各占一位。e为大写时,输出大写E
%g	g(或G)格式,只是为了使输出宽度最小,由系统决定采用%f还是%e格式

【例 2.4】 把数 -1234.6789 和 -4321.9876 按浮点数格式输出。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
float x = -1234.6789;
```

```
/* 定义变量 x 并赋值 */
```

```
double y = -4321.9876;
```

```
/* 定义变量 y 并赋值 */
```

```
printf("x = %f\n",x);
```

```
//x 的输出格式
```

```
printf("x = %12.2f\n",x);
```

```
printf("x = %-12.2f\n",x);
```

```
printf("x = %2.2f\n",x);
```

```
printf("x = %-2.2f\n",x);
```

```
printf("x = %e\n",x);
```

```
//x 的 e 格式输出
```

```
printf("x = %12.2e\n",x);
```

```
printf("x = %-12.2E\n",x);
```

```
printf("x = %2.2e\n",x);
```

```
printf("x = %-2.2E\n",x);
```

```
printf("y = %f\n",y);
```

```
//y 的输出格式
```

```
printf("y = %12.2f\n",y);
```

```
printf("y = %-12.2f\n",y);
```

```
printf("y = %2.2f\n",y);
```

```
printf("y = %-2.2f\n",y);
```

```
printf("y = %le\n",x);
```

```
//y 的 e 格式输出
```

```
printf("y = %12.2le\n",y);
```

```
printf("y = %-12.2le\n",y);
```

```
printf("y = %2.2le\n",y);
```

```
printf("y = %-2.2le\n",y);
```

程序运行结果如图 2.6 所示。

2.2.4 字符型数据

1. 字符常量

字符常量用单引号括起来,如'3'、'a'、'B'等。每个字符占一个字节。在计算机中,字符按

ASCII 码存放。ASCII 码是美国信息标准交换码 (American Standard Code for Information Intechange)。标准的 ASCII 码用 7 位二进制数对所有的英文字符进行编码, 每一个字符都对应一个 7 位二进制数, 从 ASCII 码表中可以查到各个字符对应的 ASCII 值, 见附录一。上述 4 个字符对应的 ASCII 值分别为 51、97、66、32 等, 因此字符也可以参加整型运算。

常用字符 ASCII 规律如下:

- (1) 英文字母 'A'、'B'、……、'Z' 的 ASCII 值分别为 65、66、……、90;
- (2) 英文字母 'a'、'b'、……、'z' 的 ASCII 值分别为 97、98、……、122;
- (3) 数字 '0'、'1'、……、'9' 的 ASCII 值分别为 48、49、……、57;
- (4) 回车的 ASCII 值为 13。

由于字符用单引号包围, 单引号“'”作为字符就不好表示, 在 C 中这些不好表示的字符可在字符前面加上反斜杠“\”加以区分, 如“\'”表示单引号。带反斜杠“\”表示的字符叫转义字符。常用的转义字符如表 2.4 所示。

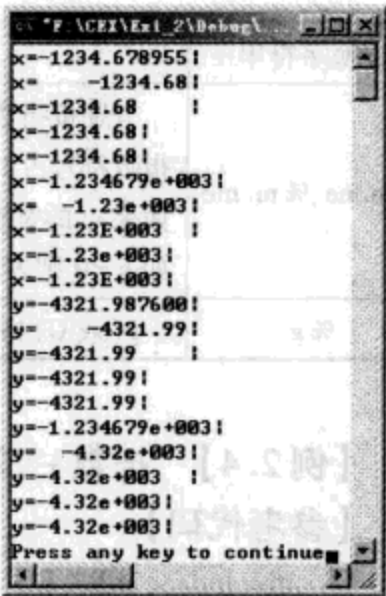


图 2.6 例 2.4 运行结果

表 2.4 常用的转义字符表

字符形式	意义	ASCII 值	字符形式	意义	ASCII 值
\b	退格	8	\'	单引号	39
\f	换页	12	\\	反斜杠	92
\n	换行	10	\v	垂直制表	11
\r	回车	13	\a	响铃	7
\t	水平制表	9	\?	问号	63
\"	双引号	34	%%	百分号	37

字符也可用八进制形式表示。例如:

八进制数 \102

它与十六进制数“\x42”都表示字母 B。

2. 字符变量

字符变量用来存放字符常量, 即只能存放单个字符, 在内存中占 1 个字节的存储空间, 用 char 来定义。字符变量定义一般形式为:

char <变量名>;

例如:

char c1, c2;

char ch1 = 'A', ch2 = 'b';

//c1, c2 表示字符变量

//ch1, ch2 表示字符变量, 并分别赋初值 'A', 'b'

3. 字符的输入/输出

字符的输入/输出在输入函数 scanf() 及输出函数 printf() 中使用。字符的输入/输出规则如表 2.5 所示。

表 2.5 字符在 scanf() 和 printf() 函数中常用的转换字符串列表

转换字符串	规则说明
%c	表示输入/输出一个字符
%wc	表示输出一个字符,宽度为 w,如果 w>0 右对齐,w<0 则左对齐
%d,%x,%o	把一个字符当作一个字节的数据输出,分别表示以十进制、十六进制和八进制数输出

【例 2.5】 输入一个字符,按字符指定的格式输出。

【解题思路】

假如定义一个变量 ch 用来存放一个字符,输入一个字符则要用格式输入函数 scanf() 按字符格式输入,即 scanf("%c",&ch),千万要注意输入格式的要求,要求什么格式就必须以什么格式输入数据。

【参考代码】

```
#include <stdio.h>
void main()
{
    char ch;
    printf("请输入一个字符:");
    scanf("%c",&ch);
    fflush(stdin);
    printf("ch 表示的字符为:%c\n",ch);
    printf("ch 表示的十进制数为:%d\n",ch);
    printf("ch 表示的十六进制数为:0x%x\n",ch);
    printf("ch 表示的八进制数为:0%o\n",ch);
}
```

程序运行结果如图 2.7 所示。

【注意点】

(1) “scanf("%c",&ch);”语句中“%c”表示输入数据以字符格式输入,“&ch”表示输入的字符存入在变量 ch 的内存地址单元,特别不能丢掉“&”符号,它是一个地址运算符。如果输入语句为“scanf("%d",&ch);”,则要输入一个十进制整数;如果输入语句为“scanf("%f",&ch);”,则要输入一个单精度实数;如果输入语句为“scanf("a=%d,b=%f\n",&a,&b);”,则要输入一个十进制整数和一个 float 型实数,并且“a=”、“b=”、“,”和“\n”一个都不能少,所以一般在编程时,输入函数中输入数据尽量不用原样输入的字符,以免出错。

(2) “fflush(stdin);”语句表示清除键盘缓冲区,即执行“scanf("%c",&ch);”语句时, ch 接收一个字符后,键盘缓冲区立即清空。这在以后的学习中特别重要。

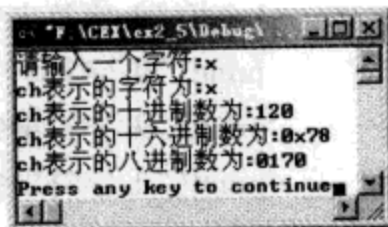


图 2.7 例 2.5 运行结果

2.2.5 字符串常量

1. 字符串常量

用双引号括起来的零个、一个或多个字符序列(包括空格),如“abcdef”,“a b c”,“ ”,“%”

a% d"都是合法的字符串常量。

特别注意:字符串常量在内存中存储时,依次存放的是串中每个字符的 ASCII 值和字符串结束标记“\0”。所以,字符串在内存中占用字符数 +1 的存储空间。例如,"abcdef"在内存中占 6 +1 个字节,在书写字符串时不必加“\0”结束标志,因为“\0”是系统自动加上的。

另外,C 语言中没有字符串变量,不能将一个字符串常量赋给一个字符串变量。如:

“char ch1 = "abcdef";”这是错误的。要想存放一个字符串,必须使用结构数据类型的数组或指针表示。

2. 字符串的输入/输出

字符串的输入/输出在输入函数 scanf() 及输出函数 printf() 中使用。字符串的输入/输出规则如表 2.6 所示。

表 2.6 字符串在 scanf() 和 printf() 函数中常用的转换字符串列表

转换字符串	规则说明
%s	表示输入/输出一个字符串
%ws	表示输出一个字符串,宽度为 w,如果 w > 0 右对齐,w < 0 则左对齐。如果 w 的宽度小于实际字符串占的位数,则按实际宽度输出
%w.ns	以宽度 w 输出字符串 s 左边的 n 个字符

【例 2.6】格式化输出一串字符。

【参考代码】

```
#include <stdio.h>
void main()
{
    printf("ch 字符串%%s 格式:%s\n", "I am a student"); // %s 格式输出
    printf("ch 字符串%%15s 格式:%15s\n", "I am a student"); // %15s 格式输出
    printf("ch 字符串%%-15s 格式:%-15s\n", "I am a student"); // %-15s 格式输出
    printf("ch 字符串%%5s 格式:%5s\n", "I am a student"); // %5s 格式输出
    printf("ch 字符串%%-5s 格式:%-5s\n", "I am a student"); // %-5s 格式输出
}
```

程序运行结果如图 2.8 所示。

【注意点】

(1)“I am a student”字符串共 14 个字符,但在内存中占有 15 个字节,因为字符串有一“\0”结束标志,“\0”表示 ASCII 值为 0 的字符。

(2)printf("ch 字符串%%-5s 格式:%-5s\n",ch) 语句中输出宽度不足,以实际字符串占有宽度输出。

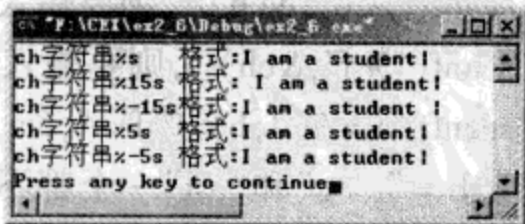


图 2.8 例 2.6 运行结果

2.3 数据类型转换

1. 自动转换

整型、单精度型、双精度型、字符型数据可以混合运算，但在运算时，不同类型的数据须先转换成同一类型，转换规则如图 2.9 所示。

即字符型向整型转换，整型向实型转换，短整型向长整型转换，无符号向有符号型转换。

例如： $5/2 + 3 * 4.5 + 3$ 的运算结果为 18.5，而不是 19。

因为 $5/2$ 结果为 2， $3 * 4.5$ 结果为 13.5，故之和为 18.5。

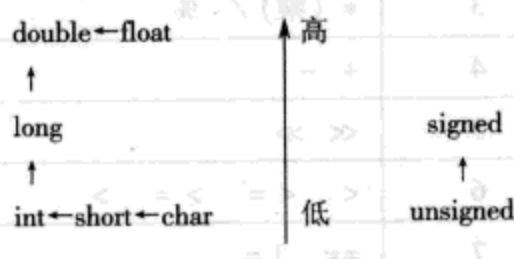


图 2.9 数据类型自动转换

2. 强制转换

强制转换的一般格式：

(目标类型) 表达式

其功能是将表达式类型转换为目标类型。

例如：

```
float a = 3.5;
```

```
int i;
```

```
i = (int)a;
```

则 i 的值为 3。

【例 2.7】强制类型转换。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{ int n;
```

```
float x = 12.64;
```

```
char ch;
```

```
n = (int)x;
```

```
ch = (char)(n + 'a');
```

```
printf("n = %d, ch = '%c'\n", n, ch);
```

```
printf("[x] = %d\n", (int)x);
```

//[x]表示 x 的取整

程序运行结果如图 2.10 所示。

2.4 运算符及表达式

C 语言提供了多种运算符，按其功能分为算术运算符、赋值运算符、自增/自减运算符、关系运算符、逻辑运算符、取地址运算符及位运算符等。

C 语言运算符如表 2.7 所示。

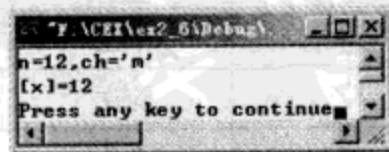


图 2.10 例 2.7 运行结果

表 2.7 C 语言运算符及结合方向

优先级	运算符	结合方向
1	() [] -> . (分量运算符)	从左至右
2	! ~ ++ -- - * (指针) & sizeof(type)	从右至左
3	* (乘) / %	从左至右
4	+ -	从左至右
5	<< >>	从左至右
6	< <= >= >	从左至右
7	== !=	从左至右
8	&	从左至右
9	^(位异或)	从左至右
10		从左至右
11	&&	从左至右
12		从左至右
13	?:(条件运算符)	从左至右
14	= += -= *= /= %= &= ^= = >>= <<=	从右至左
15	,(逗号运算符)	从左至右

C 语言表达式是由运算符、常量、变量和函数按一定的规则构成的式子。不管是简单的还是复杂的,每个表达式都会有一个值,该值称为表达式的值。

表达式中的变量使用在表达式之前必须赋值才能得出表达式的值。

2.4.1 算术运算符与表达式

1. 基本的算术运算符

算术运算符最常用的是加、减、乘、除四则运算,分别用 +、-、*、/来表示,此外,还有对整数进行的求余运算及整数除法等运算,如表 2.8 所示。

表 2.8 算术运算符

运算符	运算说明	适用类型	举例
+	加法	实数、整数	1.2 + 2.3 = 3.5; 12 + 23 = 35
-	减法	实数、整数	1.2 - 2.3 = -1.1; 12 - 23 = -11
*	乘法	实数、整数	1.2 * 2.3 = 2.76; 12 * 23 = 276
/	除法	实数、整数	1.2 / 2.3 = 0.522; 12 / 23 = 0
%	求余	整数	12 % 23 = 12
++	自加	实数、整数	a++; ++a
--	自减	实数、整数	a--; --a

2. 算术表达式

算术表达式是由算术运算符和括号连接起来的式子。

以下是算术表达式的例子：

```
x * y
(a + 2) / b
(x + y) * 5 - (a - b) / 12
i ++
sin(x) + sin(y) + 5
(++i) + (j--) + (++k)
```

3. 自加/自减运算符与表达式

自增1运算符记为“++”，其功能是使变量的值自增1。

自减1运算符记为“--”，其功能是使变量的值自减1。

自增1、自减1运算符均为单目运算，都具有右结合性。可有以下几种形式：

```
++i    /* i 自增1后再参与其他运算 */
--i    /* i 自减1后再参与其他运算 */
i++    /* i 参与运算后，i 的值再自增1 */
i--    /* i 参与运算后，i 的值再自减1 */
```

在理解和使用上容易出错的是 $i++$ 和 $i--$ 。特别是当它们出现在较复杂的表达式或语句中时，常常难于弄清，因此应仔细分析。

【例 2.8】 自增/自减运算符运算。

【解题思路】

Visual C++6.0 与 Turbo 2.0 不同：只有一个 $++i$ 时两种系统相同； $(++i) + (++i)$ 在 Visual C++6.0 中是 $i+1$ 后再 $i+1$ ，这相当于 $i=i+2$ ，然后两个 i 相加，如初值 $i=3$ ，则 $(++i) + (++i)$ 为两个 5 相加； $(++i) + (++i) + (++i)$ 则前面的 $(++i) + (++i)$ 是两个 5 相加，第 3 个 $++i$ 则是 6，所以当 i 的初值为 3 时， $(++i) + (++i) + (++i) = 5 + 5 + 6$ ；依此类推， $(++i) + (++i) + (++i) + (++i) = 5 + 5 + 6 + 7 = 23$ 。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int i=0,j;
```

```
j=++i;
```

```
printf("%2d,%2d\n",i,j); //输出 1,1
```

```
i=0;
```

```
j=(++i)+(++i);
```

```
printf("%2d,%2d\n",i,j); //输出 2,4
```

```
i=0;
```

```
j=(++i)+(++i)+(++i);
```

```
printf("%2d,%2d\n",i,j); //输出 3,7,其中 7=2+2+3
```

```
i=0;
```

```
j=(++i)+(++i)+(++i)+(++i);
```

```

printf("%2d,%2d\n",i,j);          //输出 4,11,其中 11=2+2+3+4
i=0;
j=i--;
printf("%d,%d\n",i,j);            //输出 -1,0
i=0;
j=(i--)+(i--);
printf("%d,%d\n",i,j);            //输出 -2,0
i=0;
j=(i--)+(i--)+(i--);
printf("%d,%d\n",i,j);            //输出 -3,0
i=5;
i=5;
printf("%d,%d,%d,%d\n",(++i)+(++i)+(++i)+(++i),(++i)+(++i)+(++i),
(++i)+(++i),++i);                //输出 55,31,16,6,输出从右至左原则,其中 16=8+8
                                   //31=10+10+11,55=13+13+14+15
}

```

程序运行结果如图 2.11 所示。

【注意点】

- (1) 计算 $++i, i++, --i, i--$ 的混合运算时千万要小心。
- (2) 不必过分追究 $++i, i++, --i, i--$ 繁杂的混合运算。

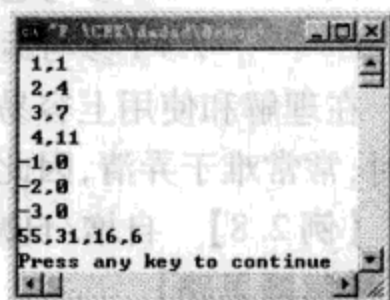


图 2.11 例 2.8 运行结果

2.4.2 赋值运算符与表达式

1. 简单赋值运算符和赋值表达式

简单赋值运算符记为“ $=$ ”。由“ $=$ ”连接的表达式称为赋值表达式。其一般形式为：

变量 = 表达式

例如：

$a = b * 5 + 3$

$f = \sin(x) + \cos(y)$

$y = i++ + --y$

赋值表达式的功能是计算表达式的值，然后赋给左边的变量。赋值运算符具有右结合性，因此 $a = b = c = 5$ 可理解为 $a = (b = (c = 5))$ 。

在 C 语言中凡是表达式可以出现的地方均可使用赋值表达式。如 $x = (a = 4) * (b = 5)$ 是合法的，它表示 4 赋给 a, 5 赋给 b, 再相乘得积 20 赋给 x, 因此 x 等于 20。

此处应注意赋值表达式与赋值语句的区别，在表达式末尾加上分号“ $;$ ”就构成赋值语句。如 $a = (b = (c = 5))$ 为赋值表达式，而“ $a = (b = (c = 5));$ ”则为赋值语句。

特别注意：

- (1) 在变量定义时，语句“ $\text{int } a = b = c = 5;$ ”是不合法的。
- (2) 在赋值表达式末尾加上分号“ $;$ ”不一定构成赋值语句。例如：

$\text{if}(a = 3) \{ \dots \}$

在 $a=3$ 后加上“;”，这个语句就是错误的语句，更不能说是赋值语句了。

2. 复合赋值运算符和赋值表达式

在赋值符“=”之前加上其他双目运算符可构成复合赋值符。C语言规定可以使用10种复合赋值运算符，其中与算术运算有关的复合运算符是： $+=$ ， $-=$ ， $*=$ ， $/=$ ， $\%=$ ， $<<=$ ， $>>=$ ， $\&=$ ， $\^{}=$ ， $|=$ （注意：两个符号之间不可以留有空格）。

构成复合赋值表达式的一般形式为

变量 双目运算符 = 表达式

它等效于

变量 = 变量 双目运算符 表达式

例如：

$a+=3$ 等价于 $a=a+3$

$x-=y+2$ 等价于 $x=x-(y+2)$

$x\%=y$ 等价于 $x=x\%y$

3. 赋值运算中的类型转换

在赋值运算中，只有在赋值运算符右侧的表达式类型与左侧的变量类型完全一致时，赋值操作才能进行，如果赋值运算符两边的数据类型不一致，系统将自动进行类型转换，即把赋值号右边的类型换成左边的类型，具有规定如下。

(1) 实型赋予整型，舍去小数部分。

(2) 整型赋予实型，数值不变，但将以浮点形式存放，即增加小数部分（小数部分的值为0）。

(3) 字符型赋予整型，由于字符为一个字节，而整型为两个字节，因此将字节的ASCII码值存放到整型量的低8位中，高8位为0。

(4) 整型赋予字符型，只把低8位赋予字符量。

【例2.9】 赋值运算的类型转换。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int a,b,c=123;
```

```
float x,y=5.6;
```

```
char ch1='a',ch2;
```

```
printf("y=%f,c=%10d,c=%3d,ch1=\'%c\'\n",y,c,c,ch1);
```

```
a=y;
```

//实型赋予整型

```
x=c;
```

//整型赋予实型

```
b=ch1;
```

//字符型赋予整型

```
ch2=c;
```

//整型赋予字符型

```
printf("a=%8d,x=%f,b=%3d,ch2=\'%c\'\n",a,x,b,ch2);
```

程序运行结果如图2.12所示。

2.4.3 逗号运算符

在 C 语言中可以用逗号“,”把若干个独立的表达式连接起来,构成逗号表达式。逗号运算符是 C 语言中优先级最低的运算符,结合规则为自左结合。其一般形式如下:

表达式 1,表达式 2,……表达式 n

作用是:依次求表达式 1 的值,表达式 2 的值,……表达式 n 的值,且表达式 n 的值为整个逗号表达式的值。例如:

$x = 1 + 2, 3 * 4$ //x 的值为 3,不是 7

注意:赋值运算符优先于逗号运算符, $x = 1 + 2$ 为赋值表达式, $3 * 4$ 为算术表达式。例如:

$x = (1 + 2, 3 * 4)$ //x 的值为 12

$x = (x = 1 + 2, x * 4)$ //x 的值为 12,先 $x = 1 + 2$,再 $x * 4 = 12$,后 $x = 12$

【例 2.10】逗号表达式输出。

【参考代码】

```
#include <stdio.h>
main()
```

```
{
```

```
    int a,b,c;
```

```
    printf("输入两个数为:");
```

```
    scanf("%d%d",&a,&b);
```

```
    printf("输出第 1 个逗号表达式中 printf() 输出的结果为:");
```

```
    c * = (a ++ , b ++ , c = a + b, printf("%d\n",c), c + = a * b);
```

```
    printf("输出第 2 个逗号表达式为:");
```

```
    printf("%d\n",(a,b,c));
```

```
}
```

程序运行结果如图 2.13 所示。

【注意点】

(1) “ $c * = (a ++ , b ++ , c = a + b, printf("%d\n",c), c + = a * b);$ ”语句中执行顺序为:① $a ++$ 后 $a = 4$;② $b ++$ 后 $b = 5$;③ $c = a + b, c = 9$;④ $printf("%d\n",c)$, 输出 9;⑤ $c + = a * b, c = 29$;⑥ $c * = (a ++ , b ++ , c = a + b, printf("%d\n",c), c + = a * b)$ 即 $c = c * c, c = 841$ 。

(2) 执行“ $printf("%d\n", (a,b,c));$ ”语句后输出 841。请仔细阅读本程序。

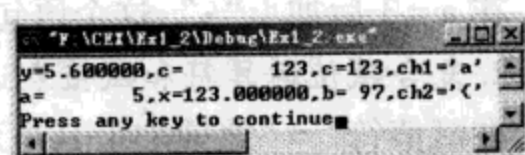


图 2.12 例 2.9 运行结果

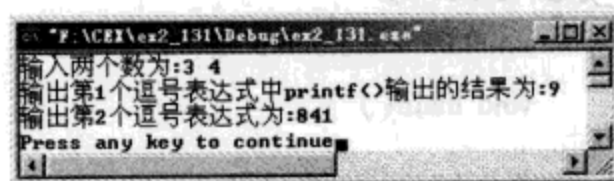


图 2.13 例 2.10 运行结果

本章小结

(1) C 语言提供了丰富的数据类型,基本的数据类型有整型、实型和字符型。

(2) 常量是指在程序运行过程中不能改变的量。常量有整型常量、实型常量、字符型常量和字符串常量,还可以用符号常量。

(3) 变量是指在程序运行过程中可以改变的量。变量实际上代表内存的某存储空间。

(4) C 语言的表达式非常丰富,有算术表达式、赋值表达式、逗号表达式等。

(5) 本章重点介绍的是常量、变量的概念,数据类型的概念、输出格式及运算符和表达式等。

习题二

一、选择题

1. 以下选项中不能作为 C 语言合法常量的是()。

- A) 'cd' B) 0.1e+6 C) "\a" D) '\011'

2. 以下选项中正确的实型常量是()。

- A) 0 B) 3.1415 C) 0.329 * 10² D) .871

3. 以下选项中不正确的实型常量是()。

- A) 2.607E-1 B) 0.8103e2 C) -77.77 D) 456e-2

4. 有以下程序,其中 %u 表示按无符号整数输出。

```
#include <stdio.h>
```

```
main()
```

```
{ unsigned int x = 0xFFFF;
```

```
printf("%u\n", x);
```

```
}
```

程序运行后的输出结果是()。

- A) -1 B) 65535 C) 32767 D) 0xFFFF

5. 已知大写字母 A 的 ASCII 码是 65,小写字母 a 的 ASCII 码是 97,以下不能将变量 c 中大写字母转换为对应小写字母的语句是()。

- A) c = (c - 'A') % 26 + 'a' B) c = c + 32
C) c = c - 'A' + 'a' D) c = ('A' + c) % 26 - 'a';

6. C 语言中运算对象必须是整型的运算符是()。

- A) % B) / C) ! D) * *

7. 可在 C 程序中用作用户变量标识符的一组标识符是()。

- A) void define WORD B) as_b3_123 If
C) For -abc case D) 2c DO SIG

8. 以下选项中,合法的一组 C 语言数值常量是()。

- A) 028 B) 12. C) .177 D) 0x8A
.5e-3 0Xa23 4e1.5 10,000
-0xf 4.5e0 Oabc 3.e5

9. 若变量已正确定义并赋值,符合 C 语言语法的表达式是()。

- A) a = a + 7; B) a = 7 + b + c, a++ C) int(12.3%4) D) a = a + 7 = c + b

10. 以下叙述中正确的是()。

- A) a 是实型变量,C 允许进行以下赋值 a = 10,因此可以这样说:实型变量允许赋值整型

值

B) 在赋值表达式中, 赋值号左边既可以是变量也可以是任意表达式

C) 执行表达式 $a=b$ 后, 在内存中 a 和 b 存储单元中的原有值都将被改变, a 的值已由原值改变为 b 的值, b 的值由原值变为 0

D) 已有 $a=3, b=5$ 。当执行了表达式 $a=b, b=a$ 之后, 已使 a 中的值为 5, b 中的值为 3

11. 以下关于 long、int 和 short 类型数据占用内存大小的叙述中正确的是()。

A) 均占 4 个字节

B) 根据数据的大小来决定所占内存的字节数

C) 由用户自己定义

D) 由 C 语言编译系统决定

12. C 源程序中不能表示的数制是()。

A) 二进制

B) 八进制

C) 十进制

D) 十六进制

13. 不合法的八进制数是()。

A) 0

B) 028

C) 077

D) 01

14. 不合法的十六进制数是()。

A) 0xff

B) 0Xabc

C) 0x11

D) 0x19

15. 已知字符 'A' 的 ASCII 代码值是 65, 字符变量 $c1$ 的值是 'A', $c2$ 的值是 'D'。执行语句 "printf("%d,%d", $c1, c2-2$);" 后, 输出结果是()。

A) A,B

B) A,68

C) 65,66

D) 65,68

16. 设变量已正确定义并赋值, 以下正确的表达式是()。

A) $x=y*5=x+z$ B) $\text{int}(15.8\%5)$ C) $x=y+z+5, ++y$ D) $x=25\%5.0$

17. 若变量均已正确定义并赋值, 以下合法的 C 语言赋值语句是()。

A) $x=y==5;$

B) $x=n\%2.5;$

C) $x+n=i;$

D) $x=5=4+1;$

18. 有以下程序段:

```
char ch;
```

```
int k;
```

```
ch='a';
```

```
k=12;
```

```
printf("%c,%d,", ch, ch, k);
```

```
printf("k=%d\n", k);
```

已知字符 a 的 ASCII 十进制代码为 97, 则执行上述程序段后输出结果是()。

A) 因变量类型与格式描述符的类型不匹配输出无定值

B) 输出项与格式描述符个数不符, 输出为零值或不定值

C) $a, 97, 12k=12$

D) $a, 97, k=12$

19. 设有定义 "int $k=0$;", 以下选项的四个表达式中与其他三个表达式的值不相同的是()。

A) $k++$

B) $k+=1$

C) $++k$

D) $k+1$

20. 以下正确的字符串常量是()。

- A) "\\\" B) 'abc' C) OlympicGames D) ""

21. 下列程序的运行结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int x,y,z;
```

```
    x=0;
```

```
    y=-3;
```

```
    z=-2;
```

```
    x+=-z---y;
```

//相当于 $x+ = -(z--)-y$; 自左至右

```
    printf("x=%d\n",x);
```

```
}
```

A) x=4

B) x=0

C) x=2

D) x=3

22. 已知“int a; float b;”所用的 scanf 调用语句格式为:

```
scanf("a//%d,b=%f",&a,&b);
```

为了将数据 3 和 25.08 分别赋给 x 和 y, 正确的输入应当是()。

A) 3,25.08 <Enter>

B) a=3,b=25.08 <Enter>

C) a//3,b=25.08 <Enter>

D) a//3 <Enter> b=25.08 <Enter>

二、填空题

1. 已知定义“char c=' '; int a=1,b;”(此处 c 初值为空格), 执行“b=!c&&a;”后 b 的值是_____。

2. 设变量已正确定义为整型, 则表达式 $n=i=2, ++i, i++$ 的值为_____。

3. 若 k 为 int 整型变量且赋值 11, 则运算 $k++$ 后表达式的值为_____, 变量的值为_____。

4. 若 x 为 double 型变量, 则运算 $x=2.1, ++x$ 后表达式的值为_____, 变量的值为_____。

5. C 语言中的标识符可分为_____、_____和预定义标识符三类。

6. 在 C 语言程序中, 用关键字_____定义基本整型变量, 用关键字_____定义单精度实型变量, 用关键字_____定义双精度实型变量。

7. 表达式 $3.5 + 1/2$ 的计算结果是_____。

8. 当计算机用两个字节存放一个整数时, 其中能存放的最大(十进制)整数是_____, 最小(十进制)整数是_____, 它们的二进制数的形式是_____。

9. 在 C 语言中整数可用_____进制数、_____进制数和_____进制数三种数制表示。

10. 以下程序的输出结果是_____。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int x=10,y=10;
```

```
    printf("%d %d\n",x--,--y);
```

```
}
```


11. 设 $x=2.5$, $a=7$, $y=4.7$, 表达式 $x+a\%3*(x+y)\%2/4$ 的值为_____。

12. 设 $a=2$, $b=3$, $x=3.5$, $y=3.5$, 表达式 $(\text{float})(a+b)/2 + (\text{int})x\%(\text{int})y$ 的值为_____。

13. 分析下列程序执行结果_____。

```
#include <stdio.h>
void main()
{
    int x=10,y=9;
    int a,b,c;
    a=(--x==y++)? --x: ++y;
    b=x++;
    c=y;
    printf("%d,%d,%d\n",a,b,c);
}
```

14. 输出下列程序结果_____。

```
#include <stdio.h>
void main()
{
    int a=2,b=4,c=6,x,y;
    y=(x=a+b),(b+c);
    printf("x=%d,y=%d\n",x,y);
    a=2,b=4,c=6;
    y=((x=a+b),(b+c));
    printf("x=%d,y=%d\n",x,y);
    a=2,b=4,c=6;
    y=x=a+b,b+c;
    printf("x=%d,y=%d\n",x,y);
    a=2,b=4,c=6;
    y=x=(a+b,b+c);
    printf("x=%d,y=%d\n",x,y);
}
```

/* 等价于“y=x=a+b,b+c;” */

/* 逗号运算符优先级最低 */

三、编程题

1. 若 $a=3$, $b=4$, $c=5$, $x=1.2$, $y=2.4$, $z=3.6$, $u=51274$, $n=7654321$, $c1='a'$, $c2='b'$, 想得到如图 2.14 所示的输出格式和结果, 请编写程序(包括定义变量类型和设计输出)。

2. 编写一个程序, 用于接收用户输入的两个数(可以是整数也可以是小数), 对其执行加、减、乘、除及求余运算, 然后以格式化方式显示计算结果, 要求小数点保留两位有效数字。

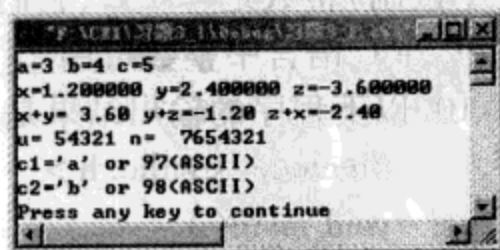


图 2.14 习题要求的结果

第3章 基本输入/输出函数

本章介绍 C 语言的基本输入/输出函数,兼介绍 C 语言的顺序结构程序设计。通过本章的学习,读者应该掌握如下内容:

- (1) 格式化输入 `scanf()` 与输出 `printf()` 函数的常用操作;
- (2) 字符输入 `getchar()` 与输出 `putchar()` 函数的常用操作。

3.1 输入/输出函数

3.1.1 格式化输入函数 `scanf()`

为了使计算机完成处理功能,用户需要输入原始数据,经过有效处理后,计算机需要将有用的信息显示给用户。C 语言中提供了 `scanf()` 和 `printf()` 两个函数来分别实现输入/输出功能。`scanf()` 和 `printf()` 两个函数是 C 的标准库函数,它们的函数原型在头文件“`stdio.h`”中定义,因此在 Visual C++ 6.0 中程序最前面应该包括:

```
#include <stdio.h>
```

这行的作用是告诉编译程序,在本程序中使用了 C 标准库里的输入/输出函数,要求编译程序正确处理这些函数的使用。

`scanf()` 函数的一般形式为:

```
scanf("格式描述串",变量地址列表);
```

在 `scanf()` 中输入的信息要遵循如下规则。

(1) “变量地址列表”是由一个或多个变量地址组成,即在变量名前加地址操作符“&”,这是初学者易忽略的一个问题。当变量地址有多个时,各变量地址之间用逗号“,”分隔。“&”不能作用于表达式,因为表达式没有地址,只有值。

(2) “格式描述串”规定数据的输入格式必须用双引号括起,其内容由转换字符串(以“%”开头,以转换字符结束,中间可有指明数据宽度的正整数等),也可以包含转义序列(以“\”标识)。

(3) 输入数据时,普通字符必须按原样输入。

(4) `scanf()` 函数输入实数时不能规定精度。

(5) 变量地址列表中,变量的个数要与格式描述串中格式描述符号的个数相等,否则出错。

(6) 变量地址列表中,变量的定义的数据类型与格式描述串中转换字符相符。

(7) 用户键入回车符后才开始从键盘缓冲区中读入数据,在此之前,用户可以修改输入的数据。

【例 3.1】 使用 `scanf()` 格式化输入。

【参考代码】

```
#include <stdio.h>
void main()
{
    char ch;
    int i;
    long l;
    float f,fe;
    double d,de;
    printf("请输入数据:");
    scanf("%c%d%ld%f%e%lf%le",&ch,&i,&l,&f,&fe,&d,&de);
    printf("ch=%c i=%d l=%ld f=%f fe=%e d=%lf de=%le\n",ch,i,l,f,fe,d,de);
}
```

程序运行结果如图 3.1 所示。

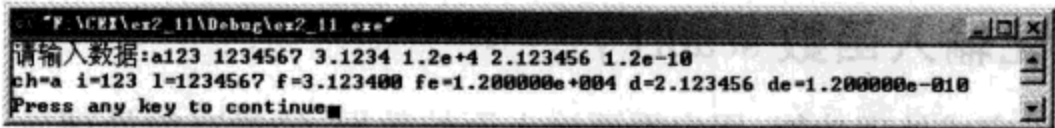


图 3.1 例 3.1 运行结果

【注意点】

- (1) 当输入的信息与对应的格式串要求不符时,输出不会得到预期结果。
- (2) 分隔输入数据信息的空白符可以为空格、制表符和回车符等。

3.1.2 格式化输出函数 printf()

printf()函数的一般形式为:

```
printf("格式描述串",变量(或表达式)列表);
```

第一个参数说明采用什么样的格式输出内容。格式描述串中可以包括各种格式转换字符串(以%开头),也可以包括转义串(以“\”标识),详见前几章中各种数据类型的输入/输出。例如%d、%ld、%f和%lf分别表示要求输入的输入项类型为int、long、float和double。第二个参数是要显示其值的参数列表。如果格式描述串中没有特殊字符串“%”,那么该输出语句就不该有变量(或表达式)列表,也不需要表示分隔的逗号(特殊情况“%%”除外),这是使用printf()最简单的形式。这种形式的输出语句的作用就是输出格式描述串本身。例如,“printf(“This is a c program.”);”语句的执行结果是输出:

This is a c program.

格式描述串的转换字符串在前一章数据类型中几乎全部介绍过,现归纳如表 3.1 所示。

表 3.1 C 语言中的转换字符串(部分)

转换字符串	规则说明	转换字符串	规则说明
%c	输出字符	%g,%G	以最短长度输出 f 或 e 格式的实数
%d	输出十进制整数	%x,%X	输出十六进制整数

【注意】续表

转换字符串	规则说明	转换字符串	规则说明
%f	输出 float 实数	%o,%O	输出八进制整数
%e,%E	输出 e 格式 float 实数	%ld(或 f,e,E 等)	输出十进制长整型数(或 double、e(E) 格式的 double 型实数)
%u	输出无符号十进制整数	%wd(或 u,x,o,s 等)	输出宽度为 w 的数(或字符)
%s	输出字符串	%m.nf(或 e 等)	输出宽度为 m,小数位数为 n 的实数
%w.ns	输出宽度为 w,取字符串前 n 个字符		

【例 3.2】 写出下列程序的运行结果。

【参考代码】

```
#include <stdio.h>

void main()
{
    int a = 3,b = 4;
    float x = 54.8765,y = -789.432;
    char c = 'B';
    unsigned u = 65535;
    long n = 1234567;
    printf("输出的结果为:\n"); //提示输出行
    printf("%d%d\n",a,b);
    printf("%3d%3d\n",a,b);
    printf("%f,%f\n",x,y); //f 格式默认小数位数 6 位
    printf("%-10f,%-10f\n",x,y); //"- "表示左对齐
    printf("%8.2f,%8.2f,%4f,%4f,%3f,%3f\n",x,y,x,y,x,y);
    printf("%e,%10.2e\n",x,y); //e 格式默认小数位数 6 位,同 f 格式
    printf("%c,%d,%o,%x\n",c,c,c,c);
    printf("%ld,%lo,%x\n",n,n,n);
    printf("%u,%o,%x,%d\n",u,u,u,u);
    printf("%s,%5.3s\n","computer","COMPUTER");
}
```

程序运行结果如图 3.2 所示。

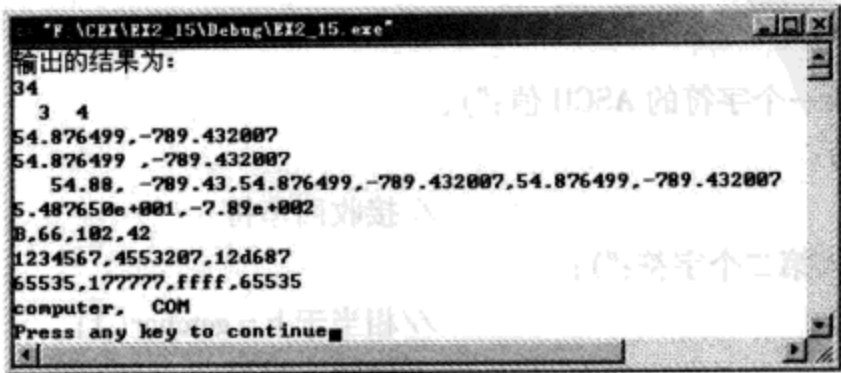


图 3.2 例 3.2 运行结果

【注意点】

printf("%s,%5.3s\n","computer","COMPUTER"); 语句中“%5.3s”,5 为输出字符串宽度,3 为取字符串前 3 个字符,右对齐。

3.2 字符输入/输出函数

3.2.1 字符输入函数 getchar()

字符输入函数 getchar()专门用于输入字符型数据。虽然使用 scanf()函数可以输入所有类型的数据,但输入字符数据时使用 getchar()更为简单明了。

getchar()的一般形式如下:

字符型变量 = getchar();

其中,字符型变量是准备接收字符型数据的。程序运行到这里时,将停下等待用户输入字符,用户输入数据完毕后按回车继续运行。

注意:如果用户输入多个字符,则只有第一个字符有效,其他字符将会丢失或被后面的语句接收。

3.2.2 字符输出函数 putchar()

字符输出函数 putchar()专门用于输出字符型数据。例如 putchar(c);是输出字符变量(或整型变量)c 的值。

putchar()的一般形式如下:

putchar(c);

注意:putchar()必须带输出项,如 c,c 可以是字符型常量、变量、表达式,但只能是单个字符而不能是字符串;c 也可以是整型常量,则该常量被看作字符的 ASCII 值,输出的是该整型常量值所对应的字符。例如,语句“putchar(65);”输出的为大写字母 A。

【例 3.3】 输出下列程序的结果。

【参考代码】

```
#include <stdio.h>
void main()
{
    int a;
    char b;
    printf("请输入第一个字符的 ASCII 值:");
    scanf("%d",&a);
    getchar(); //接收回车符
    printf("\n 请输入第二个字符:");
    scanf("%c",&b); //相当于 b = getchar();
    getchar(); //接收回车符
    printf("输出的第一个字符为:");
    putchar(a);
```

```
putchar('\n');  
printf("输出的第二个字符为:");  
putchar(b);  
putchar('\n');
```

程序运行结果如图 3.3 所示。

【注意点】

(1) 在语句“scanf("%d",&a);”接收了一个字符的 ASCII 后,如果没有“getchar();”语句来接收回车符,则语句“scanf("%c",&b);”中 b 就会接收回车符,所以要小心对待。

(2) “getchar();”语句可以用“fflush(stdin);”语句替代。“fflush(stdin);”语句表示清除键盘缓冲区。

(3) “putchar('\n');”表示换行。

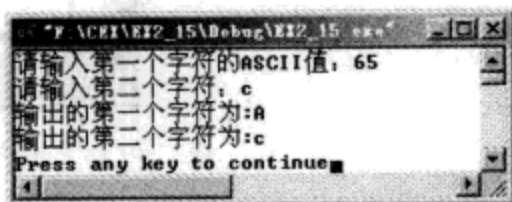


图 3.3 例 3.3 运行结果

3.3 案例应用举例

【例 3.4】 利用输入与输出语句编写“图书管理系统”的主菜单。

【参考代码】

```
#include <stdio.h>  
void main()  
{  
    char chl;  
    printf("***** 欢迎进入图书馆管理系统 *****\n");  
    printf(" *  
    printf(" *          制作人:汤承林          * \n");  
    printf(" *          联系地址:江苏 × × × 信息学院计算机科学与工程系          * \n");  
    printf(" *          * * * * *          * \n");  
    printf(" *          请你选择操作类型:          * \n");  
    printf(" *          1: < 图书信息管理 >          * \n");  
    printf(" *          2: < 借书卡管理 >          * \n");  
    printf(" *          3: < 图书借还管理 >          * \n");  
    printf(" *          0: < 退出 >          * \n");  
    printf("***** \n");  
    printf("请选择 0 -- 3:");  
    scanf("%c",&chl);  
    getchar();  
    printf("您选择的是第%c 项!\n",chl);  
}
```

程序运行结果如图 3.4 所示。

【注意点】

语句“getchar();”用于接收输入字符(0 -- 3)之后的“回车”符,以免以后的程序语句接

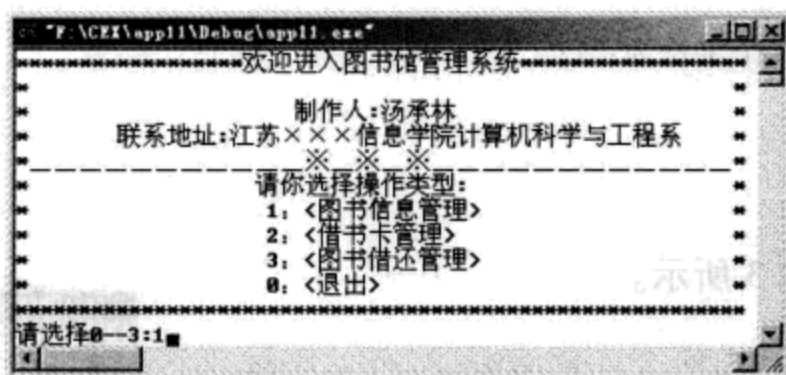


图 3.4 主菜单

收该“回车”符而使程序产生不可理解的结果。

本章小结

(1) 本章介绍了输入/输出语句 `scanf()` 和 `printf()`, `getchar()` 和 `putchar()`。重点介绍了 `scanf()` 和 `printf()` 及输出格式。

(2) 注意 `scanf()` 函数中, 格式描述串中的每个非格式控制字符必须原样输入, 所以读者在输入语句中尽量不要使用非格式控制的字符。

习题三

一、选择题

1. 以下选项中不是 C 语句的是()。

- A) `{ int i; i++; printf("%d\n", i); }` B) `};`
 C) `a = 5, c = 10` D) `{ ; }`

2. 下列程序的运行结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int x = 'f';
```

```
printf("%c\n", 'A' + (x - 'a' + 1));
```

```
}
```

- A) G B) H C) I D) J

3. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int x = 10, y = 3;
```

```
printf("%d\n", y = x/y);
```

- A)0 B)1 C)3 D)不确定的值

4. 以下程序的输出结果是()。

```
#include <stdio.h>

void main( )
{
    int m=7,n=4;
    float a=38.4,b=6.4,x;
    x=m/2+n*a/b+1/2;
    printf("%6.2f\n",x);
}
```

- A) 27.50 B) 27.00 C) 28.00 D) 28.50

5. 若变量已说明为 int 类型,要给 a、b、c 输入数据,以下正确的输入语句是()。

- A) read(a,b,c); B) scanf(" %d%d%d" ,a,b,c);
C) scanf(" %D%D%D" ,&a,%b,%c); D) scanf(" %d%d%d",&a,&b,&c);

6. 语句 `printf("a\bre\hi\y\\\bou\n");` 的输出结果是()。

- A) a\bre\'hi\'y\\bou
- B) a\bre\'hi\'y\bou)
- C) re\'hi\'you
- D) abre\'hi\'you

7. 若变量已说明为 float 类型,要通过以下赋值语句给 a 赋予 10、b 赋予 22、c 赋予 33,以下不正确的输入形式是()。

```
scanf(" %f %f %f", &a, &b, &c);
```

- A) 10 22 33 B) 10. 0, 22. 0, 33. 0 C) 10. 0 22. 0 33. 0 D) 10. 22. 33

8. 执行下列程序时,若从键盘上输入数据:

123 <回车>

678

则输出的结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
char c1,c2,c3,c4,c5,c6;
```

```
scanf("%c%c%c%c", &c1, &c2, &c3, &c4);
```

```
c5 = getchar( ) ;
```

```
c6 = getchar( ) ;
```

```
putchar( c1 );
```

```
putchar( c2 );
```

```
printf( "%c%c\n", c5, c6 );
```

- A) 1 267 B) 1 256 C) 1 278 D) 1 245

9. 若变量已正确定义, 以下程序段的输出结果是()。

```
x = 5.16894;
```

```
printf(" %f\n", (int)(x * 1000 + 0.5) / (float)1000);
```

A) 输出格式说明与输出项不匹配, 输出无定值

B) 5.17

C) 5.168

D) 5.169

10. 下列程序的输出结果是()。

```
#include <stdio.h>
```

```
void main( )
```

```
{
```

```
int x = 1, y = 2, z = 3;
```

```
printf("%d\n", z + = x > y? ++x: ++y);
```

```
}
```

A) 2

B) 3

C) 6

D) 5

11. 若有以下程序段, 其输出结果是()。

```
int a = 0, b = 0, c = 0;
```

```
c = (a - = a - 5), (a = b, b + 3);
```

```
printf(" %d, %d, %d\n", a, b, c);
```

A) 3, 0, -10

B) 0, 0, 5

C) -10, 3, -10

D) 3, 0, 3

12. 当运行以下程序时, 在键盘上从第一列开始输入 9876543210 <CR> (此处 <CR> 代表 Enter), 则程序的输出结果是()。

```
#include <stdio.h>
```

```
main( )
```

```
{
```

```
int a;
```

```
float b, c;
```

```
scanf(" %2d%3f%4f", &a, &b, &c);
```

```
printf(" \na = %d, b = %f, c = %f\n", a, b, c);
```

```
}
```

A) a = 98, b = 765, c = 4321

B) a = 10, b = 432, c = 8765

C) a = 98, b = 765.000000, c = 4321.000000

D) a = 98, b = 765.0, c = 4321.0

13. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
main( )
```

```
{
```

```
int a = 2, b = 5;
```

```
printf("a = %d, b = %d\n", a, b);
```

```
}
```

A) a = %2, b = %5

B) a = 2, b = 5

C) a = %d, b = %d

D) a = %d, b = %d

二、填空题

1. 若有以下定义,请写出以下程序段中输出语句执行后的输出结果。

```
int i = -100, j = 500;
printf("%d %d", i, j);
printf("i = %d, j = %d\n", i, j);
```

(1) _____; (2) _____。

2. 下列程序的运行结果是_____。

```
#include <stdio.h>
void main( )
{
    int x = 1, y = 2, z = 3;
    x + = y + = z;
    printf("%d\t", x < y ? y : x);
    printf("%d\t", x < y ? x ++ : y ++);
    printf("%d\t", x);
    printf("%d\t", y);
    printf("%d\t", z + = x > y ? x ++ : y ++);
    printf("%d\t%d\t", y, z);
    x = 5;
    y = z = 6;
    printf("%d\t", (z > = y == z) ? 1 : 0);
    printf("%d\n", z > y && y > = x);
}
```

3. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main( )
{
    float x = 218.82631;
    printf("%-6.2e\n", x);
}
```

4. 以下程序段的输出是_____。

```
float a = 4.1234;
printf("%6.0f\n", a);
```

5. 以下程序段的输出是_____。

```
#include <stdio.h>
void main( )
{
    int m = 0xabc, n = 0xabc;
    m + = n;
```

```
printf("%x\n", m);
```

```
}
```

6. 若变量已正确说明, 要求用以下语句给 a 赋予 3.12, 给 b 赋予 9.0, 则正确的输入形式是_____。

```
scanf("a=%f,b=%f",&a,&b);
```

7. 以下程序的输出结果是_____。

```
#include <stdio.h>
```

```
#include "math.h"
```

```
void main()
```

```
{
```

```
double a = -2.0, b = 1;
```

```
printf(" %4.0f %4.0f\n", pow(b, fabs(a)), pow(fabs(a), b));
```

```
}
```

8. 以下程序段, 要求通过 scanf() 语句给变量赋值, 然后输出变量的值。写出运行时给 k 输入 10, 给 a 输入 12.22, 给 x 输入 2.54321 时的三种可能的输入形式 _____、_____、_____。

```
int k; float a; double x;
```

```
scanf(" %d%f%lf",&k,&a,&x);
```

```
printf("k=%d,a=%f,x=%f\n",k,a,x);
```

9. 以下程序段的输出结果是_____。

```
int x=0155;
```

```
printf("x=%3d,x=%6d,x=%6o,x=%6x,x=%6u\n",x,x,x,x,x);
```

10. 以下程序段的输出结果是_____。

```
double a=521.789325;
```

```
printf("a=%8.6f,a=%8.2f,a=%14.8f,a=%14.8lf\n",a,a,a,a);
```

11. 若以下程序输入字符: a 回车 b 回车 c 回车后, 输出下列程序结果_____。

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
char a,b,c;
```

```
a=getchar();
```

```
fflush(stdin);
```

```
b=getchar();
```

```
fflush(stdin);
```

```
c=getchar();
```

```
printf("输出字符 a:");
```

```
putchar(a+1);
```

```
printf("输出字符 b:");
```

```
putchar(b+2);
```

```
printf("输出字符 c:");
putchar(c + 3);
putchar('\n');
```

12. 写出下列程序的运行结果_____。

```
#include <stdio.h>

main()
{
    char c1 = 'a', c2 = 'b';
    int i = 123, j = -54321;
    float f1 = 3.43, f2 = 1.24e + 6;
    printf("输出的结果为:");
    printf("%c%3c%d", c1, c2, i);
    printf("%-8d%12d%f", i, j, f1);
    printf("%12.2e%c%-12.3e\n%10.4s\n", f2, c2, f2, "输出的结果!");
}
```

三、编程题

- 1. 编写程序,输入两个整数:1500 和 350,求出它们的商数和余数并进行输出。
- 2. 编写程序,读入三个双精度数,求它们的平均值并保留此平均值小数点后一位数,对小数点后第二位数进行四舍五入,最后输出结果。
- 3. 设圆柱底圆半径 $r = 1.2$,圆柱高 $h = 4$,求圆的周长、面积、圆球表面积、圆球体积、圆柱体积。用 scanf() 函数输入数据,输出计算结果。输出时要有文字说明,且取小数点后两位数字。

0 或果为 1 或 1	下大	<
0 或 1 或 2 或 3 或 4	下小	>
1 或果为 1 或 2 或 3 或 4	下果为 下大	= <
1 或 2 或 3 或 4 或 5	下果为 下小	= >
1 或果为 1 或 2 或 3 或 4	下果	= =
1 或果为 1 或 2 或 3 或 4	下果不	= !



第 4 章 选择结构

选择结构是三种基本结构之一。前一章已介绍了不少小程序,这些程序大多体现了顺序结构,所以顺序结构不再单独列为一章。

选择结构的特点是程序的流程由多条支路构成,在程序的一次执行过程中,根据不同的情况,只有一条支路被选中执行,而其他支路上的语句被跳过。

4.1 关系运算和逻辑运算

4.1.1 关系运算符与表达式

1. 关系运算符

关系运算又称比较大小运算。关系运算其实就是日常生活中的比较运算。表 4.1 给出了 C 语言中使用的关系运算符。

表 4.1 C 语言中的关系运算符

运算符	运算说明	优先级	举 例
>	大于	优先级相同(高)	$1+2>3$ 结果为 0
<	小于		$1+2<3$ 结果为 0
>=	大于或等于		$1+2>=3$ 结果为 1
<=	小于或等于		$1+2<=3$ 结果为 1
==	等于	优先级相同(低)	$1+2==3$ 结果为 1
!=	不等于		$1+2!=3$ 结果为 0

C 语言中用“1”(或非 0)代表关系成立,即“真”;用“0”代表关系不成立,即“假”。

2. 关系表达式

用关系运算符将两个算术表达式或两个字符表达式连接起来的式子称为关系表达式。关系表达式的值为逻辑值。

【例 4.1】 关系表达式。

【参考代码】

```
#include <stdio.h>
void main()
{
    char ch = 'k';
    int i = 1, j = 2, k = 3;
    float x = 3e + 5, y = 0.85;
```

```
printf("%d,%d\n",'a'+4<ch,-i-2*j>=k+1);
printf("%d,%d\n",1<j<5,x-5.12<x+y);
printf("%d,%d\n",i+j+k== -2*j,k==j==i+5);
```

程序运行的结果如图 4.1 所示。

【注意点】

- (1) 字符变量是以它对应的 ASCII 值参加运算。
- (2) 对含有多个关系运算符的表达式,如 $1 < j < 5$,根据运算符的左结合性及优先级,先比较 $1 < j$,其结果为 1,再将结果“1”代入后半部分表达式,即 $1 < 5$,比较 $1 < 5$,其结果为 1。

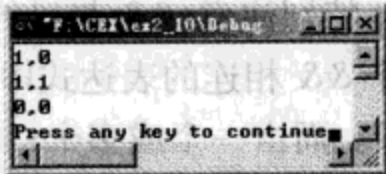


图 4.1 例 4.1 运行结果

4.1.2 逻辑运算符与表达式

1. 逻辑运算符

在实际应用中,经常会遇到非常复杂的多个条件的情况。例如,要判定某一年是否为闰年的条件应符合以下二者之一:

- (1) 能被 4 整除,但不能被 100 整除;
- (2) 能被 400 整除。

要表达这样一种条件,利用前面介绍的表达式就会遇到困难。因此,在高级语言中,需要引进逻辑运算符和逻辑表达式的概念,以表达复杂的条件。C 语言提供了 3 种逻辑运算符,如表 4.2 所示。

表 4.2 C 语言中的逻辑运算符

运算符	举例	说明
&&(与)	$x \&\&y$	二元运算,仅当 x,y 两者都为真时结果为真,否则为假
(或)	$x y$	二元运算,只要 x,y 两者有一为真时结果为真,否则为假
!(非)	$!x$	一元运算,当 x 为真时结果为假, x 为假时结果为真

在 $\&\&$ 、 $||$ 和 $!$ 这 3 种逻辑运算中, $!$ (非) 优先级最高, $\&\&$ (与) 优先级次之, $||$ (或) 优先级最低。例如 $x \&\&y || !z$, 先运算 $!z$, 之后运算 $x \&\&y$, 最后运算 $||$ 。

非(!)运算作用在 $\&\&$ 、 $||$ 及 $!$ 运算中有如下规则:

- (1) $!(x \&\&y)$ 等价于 $!x || !y$;
- (2) $!(x || y)$ 等价于 $!x \&\&!y$;
- (3) $!(!x)$ 等价于 x 。

2. 逻辑表达式

逻辑运算常常与关系运算相结合,形成逻辑运算表达式,在这种表达式中,关系运算要先于逻辑运算。例如:

```
x + y > z && x + z > y && y + z > x ;
x > y || x > z;
!x || y > z;
```

其中 $x + y > z \&\& x + z > y \&\& y + z > x$ 表示只有当 $x + y > z, x + z > y, y + z > x$ 三个条件同时成立时,结果才为真。 $x > y || x > z$ 表示 $x > y$ 与 $x > z$ 之一成立,结果为真; $!x || y > z$ 表示只要 $!x$ 与 $y > z$ 之一为真,结果就为真。

C 语言规定,结果为逻辑值的表达式值为非 0 时,表达式的结果就为“真”,否则为“假”。

特别注意:&& 与 || 的“短路”作用。C 语言中,&& 与 || 是“短路”运算符,即在一个或多个 && 相连的表达式中,只要有一个操作数为 0,就不做后面的 && 运算,整个表达式的结果为 0。而由一个或多个 || 相连的表达式中,只要遇到第一个不为 0 的操作数,就不再进行后面的 || 运算,整个表达式的值为 1。例如,对于表达式:

(条件 1)&&(条件 2)

若(条件 1)为 0,则不需要再判断(条件 2),立即可以判断整个表达式的值为 0。

4.2 if 语句

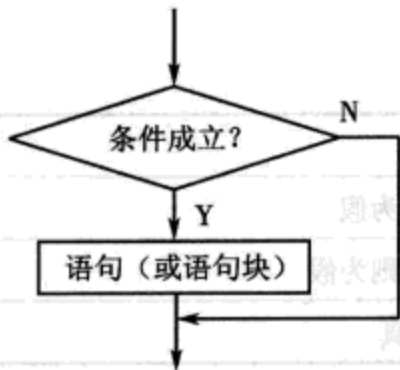
4.2.1 简单 if 语句

简单的 if 语句的一般形式为:

if(条件)

语句(或语句块);

}



它的含义是当条件满足时,执行括号内的语句(或语句块),执行完后接着执行“}”后的语句;如果条件不满足,则括号内语句不执行,转去执行“}”后面的语句,如图 4.2 所示。

注意:

(1) 语句块指的是多条语句,如果语句块只是一条语句,则

“{”和“}”可以省略。

(2) 条件两边的圆括号“(”和“)”不能少。

(3) “)”后不能有分号“;”,“}”后也不要分号“;”。

图 4.2 if 语句的执行流程

【例 4.2】 输入一个成绩,判断是否通过考试。

【参考代码】

```
#include <stdio.h>
main()
{
    int score;
    printf("请输入成绩:");          //提示输入行
    scanf("%d",&score);
    if( score >= 60) printf("通过!\n");
    if( score < 60) printf("未通过!\n");
}
```

程序执行结果如图 4.3 所示。

【例 4.3】 输入三个数,并由小到大排序。

【解题思路】

假设有三个变量 x, y, z 存放三个数,一个变量 t 作为临时存放中间数。由于 `if`(条件表达式)语句中条件满足时,需要交换两次,因而需要三条语句才能实现两个数的交换。

其方法是把最小的数放 x 中,先将 x 和 y 进行比较,如果 $x > y$,则将 x 与 y 的值进行交换;然后再用 x 与 z 进行比较,如果 $x > z$,则将 x 与 z 的值进行交换;最后再将 y 与 z 进行比较,如果 $y > z$,则将 y 与 z 的值进行交换。

【参考代码】

```
#include <stdio.h>

void main()
{
    int x, y, z, t;
    printf("请输入三个数:");
    scanf("%d%d%d", &x, &y, &z);
    printf("三个数分别为:x = %d, y = %d, z = %d\n", x, y, z);
    if (x > y)                /* 交换 x, y 的值 */
    {
        t = x;
        x = y;
        y = t;
    }
    if (x > z)                /* 交换 x, z 的值 */
    {
        t = x;
        x = z;
        z = t;
    }
    if (y > z)                /* 交换 y, z 的值 */
    {
        t = y;
        y = z;
        z = t;
    }
    printf("x, y, z 由小到大排序:x = %d, y = %d, z = %d\n", x, y, z);
}
```

程序执行结果如图 4.4 所示。

4.2.2 二分支 if 语句

二分支 if 语句的一般形式为:

`if(条件)`

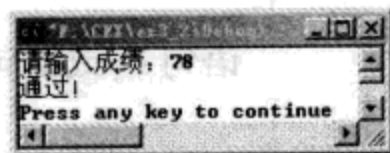


图 4.3 例 4.2 运行结果


```

    语句 1(或语句块 1);
}
else

```

```

}

```

```

    语句 2(或语句块 2);

```

它的含义是当条件满足时,执行语句 1(或语句块 1),否则执行语句 2(或语句块 2)。其程序执行流程如图 4.5 所示。

【例 4.4】 输入一个年份 year,判定 year 是否为闰年。

【解题思路】

判定 year 是否为闰年,只要判定 year 是否满足如下条件之一:

- (1) year 能被 4 整除,但不能被 100 整除;
- (2) year 能被 400 整除。

【参考代码】

```

#include <stdio.h>
void main()
{
    int year;
    printf("请输入年份:");
    scanf("%d",&year);
    if((year%4==0&&year%100!=0)|| (year%400==0))
    {
        printf("%d 是闰年!\n",year);
    }
    else
    {
        printf("%d 不是闰年!\n",year);
    }
}

```

程序运行结果如图 4.6 所示。

4.2.3 二分支 if 语句嵌套

在 if 语句中可以是语句块(或称复合语句),而在语句块中又可以嵌套另一个 if 语句,这样可以组成 if 语句的嵌套。

【例 4.5】 输入三个数求其最大者。

【解题思路】

假设有三个变量 x,y,z 存放三个数。可以采用例 4.3 的方法解决这个问题,这里我们换

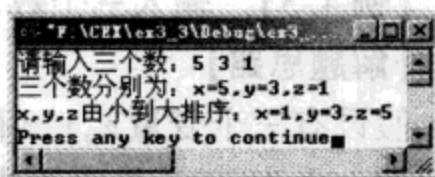


图 4.4 例 4.3 运行结果

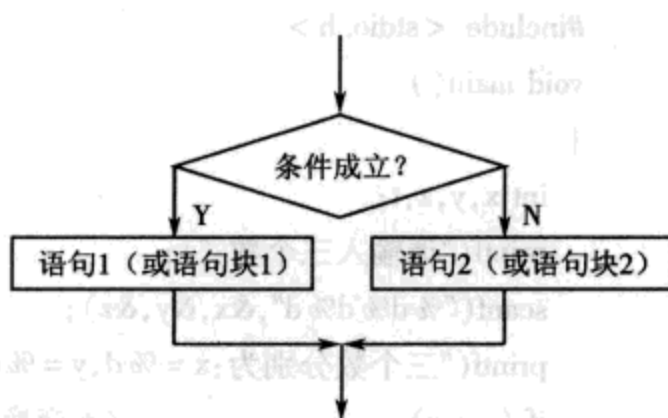


图 4.5 二分支 if 语句的执行流程

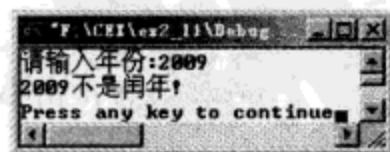


图 4.6 例 4.4 运行结果

个思路考虑:①先假设 $x < y$ 成立,再判定是否 $y < z$;②如果 $y < z$ 成立,则 z 最大;③如果 $y < z$ 不成立,则 y 最大;④如果 $x < y$ 不成立,再判定是否 $x < z$ 。

【参考代码】

```
#include <stdio.h>
void main()
{
    int x,y,z;
    printf("请输入三个数:");
    scanf("%d%d%d",&x,&y,&z);
    printf("三个数分别为:x=%d,y=%d,z=%d\n",x,y,z);
    if (x < y)
    {
        if (y < z)
            printf("最大值 z = %d\n",z);
        else
            printf("最大值 y = %d\n",y);
    }
    else
    {
        if (x < z)
            printf("最大值 z = %d\n",z);
        else
            printf("最大值 x = %d\n",x);
    }
}
```

二重if嵌套

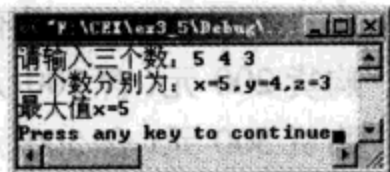
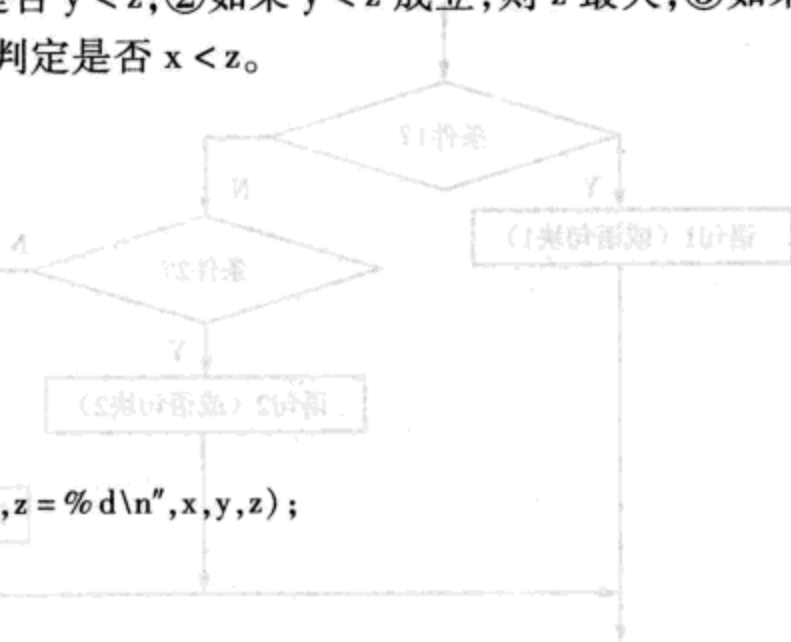


图 4.7 例 4.5 运行结果

程序运行结果如图 4.7 所示。

4.2.4 多分支 if 语句

在二分支基础上语句 2(或语句块 2)的进一步复合可实现多分支结构,这就是多分支 if 语句。其一般形式为:

```
if(条件 1)
{ 语句 1(或语句块 1); }
else
if(条件 2)
{ 语句 2(或语句块 2); }
...
if(条件 n)
{ 语句 n(或语句块 n); }
else
{ 语句 n+1(或语句块 n+1); }
```

其执行过程:其条件 1 满足则执行语句 1(或语句块 1),然后越过所有的 if 语句去执行 if 语句的下一条语句;若条件 1 不满足,则判断条件 2 是否满足,如满足则执行语句 2(或语句块 2),然后越过所有的 if 语句去执行 if 语句的下一条语句;若条件 2 不满足,则判断条件 3 是否满足,……,则执行语句 $n+1$ (或语句块 $n+1$)。其流程图如图 4.8 所示。

注意:if 与 else 配对规则。C 语言规定,else 子句总是与前面最近的一个 if 相配对。

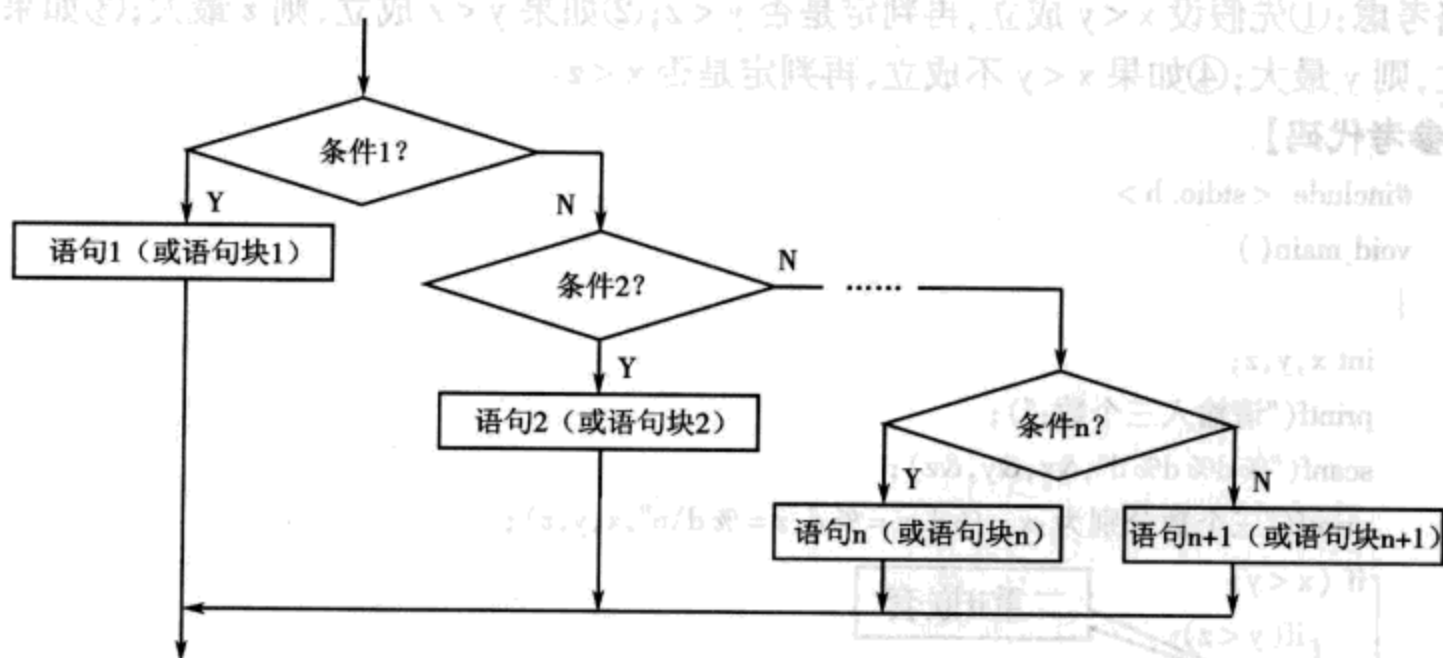


图 4.8 多支 if 语句的执行流程

【例 4.6】 商店销售货物时,常按购买货物款的多少分别给予相应的折扣,试编程计算实际应付款。

购货不足 200 元,没有折扣;

购货 200 元(含 200 元),不足 500 元,折扣 95%;

购货 500 元(含 500 元),不足 1 000 元,折扣 90%;

购货 1 000 元(含 1 000 元),不足 2 000 元,折扣 85%;

购货 2 000 元及以上,折扣 80%。

【解题思路】

设购物款为 m ,折扣为 d ,则 d 可表示为:

$$d = \begin{cases} 1 & m < 200 \\ 0.95 & 200 \leq m < 500 \\ 0.90 & 500 \leq m < 1\,000 \\ 0.85 & 1\,000 \leq m < 2\,000 \\ 0.80 & 2\,000 \leq m \end{cases}$$

利用 if 语句确定 d 的值,然后即可计算实际应付款 $\text{pay} = m * d$ 。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    float m, pay;
```

```
    printf("请输入购货款数:");
```

```
    scanf("%f", &m);
```

```
    if (m < 200)
```

```
    {
```

```
        pay = m;
```

```
    }
```

```
    else if (m >= 200 && m < 500)
```

```
    {
```

```

    pay = m * 0.95;
}
else if( m >= 500 && m < 1000)
{
    pay = m * 0.90;
}
else if( m >= 1000 && m < 2000)
{
    pay = m * 0.85;
}
else
{
    pay = m * 0.80;
}
printf("应付款 = %7.2f 元,实付款 = %7.2f 元\n",m,pay);
}

```

程序运行结果如图 4.9 所示。

4.2.5 条件运算符

条件运算符是 C 语言中唯一的三元运算符,用符号?:表示,它带有 3 个操作数,优先级见表 2.7 所示。结合方向为从左至右,运算顺序是从右至左,其一般形式为:

表达式 1? 表达式 2:表达式 3

其意义是先计算表达式 1 的结果,如果为真则计算表达式 2 的结果作为整个表达式的值,否则计算表达式 3 的结果作为整个表达式的值。

【例 4.7】 输入两个数,比较两个数的大小。

【参考代码】

```

#include <stdio.h>
void main()
{
    int x,y;           //定义变量
    printf("请输入两个整数:"); //提示输入行
    scanf("%d %d",&x,&y);
    printf("%d 和 %d 两个数最大者是:%d\n",x,y,x>y? x:y);
}

```

程序运行结果如图 4.10 所示。

4.3 switch 语句

除了 if 嵌套语句以外,switch 语句也可以实现多分支结构。

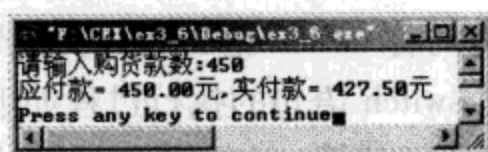


图 4.9 例 4.6 运行结果



图 4.10 例 4.7 运行结果

它将一个表达式的值与多个常量表达式的值一一比较,如果相等,则与之相应的语句被执行。

switch 语句的一般形式为:

```
switch(表达式)
{
    case 常量表达式 1:语句序列 1;[break;]
    case 常量表达式 2:语句序列 2;[break;]
    ...
    case 常量表达式 n:语句序列 n;[break;]
    [default:语句序列 n+1;]
}
```

注意:

(1) switch 语句是关键字,其后面大括号括起来的部分称为 switch 语句体。特别注意大括号不能少。

(2) switch 后的表达式不能少,表达式两边的括号也不能少,其运算结果可以是整型、字符型或枚举型(构造类型之一)等。

(3) 语句序列称为 switch 语句的子语句;switch 语句后的表达式称为开关控制表达式,所以有时称 switch 语句为开关语句;方括号内的语句可缺省,视具体情况而定。

switch 语句的执行过程如下。

(1) 计算 switch 语句后表达式的值。

(2) 逐个比较表达式的值与 case 后面的常量表达式的值是否相等。

(3) 当“表达式”的值与“常量表达式 i”的值相等时,就转去执行“语句序列 i”的各个语句,若语句序列后有 break 语句,则终止 switch 语句,然后继续执行 switch 语句体后的下一条语句。如果没有一个“常量表达式 i”的值与“表达式”的值相等,则执行语句“default”后面的“语句序列 n+1;”,然后继续执行 switch 语句体后的下一条语句。

在 switch 语句中 break 语句的使用注意点:break 语句也称间断语句,可以在各个 case 之后的语句最后加上 break 语句,每当执行到 break 语句时,立即跳出 switch 语句体。switch 语句通常总是和 break 语句联合使用的。使用 switch 语句真正起到多个分支的作用。

程序执行流程图如图 4.11 所示。

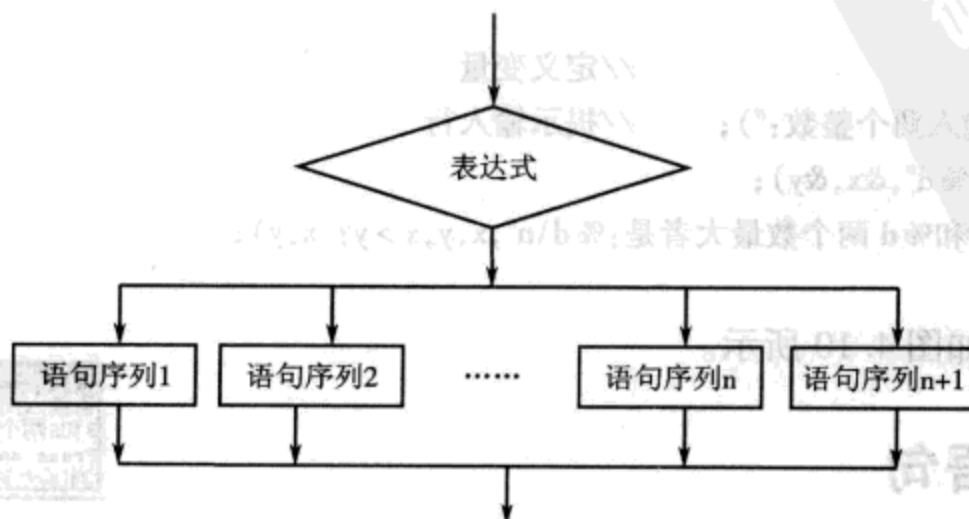


图 4.11 switch 语句的执行流程

【例 4.8】 商店销售货物时,常按购买货物款的多少分别给予相应的折扣。试采用 switch 语句编程计算实际应付款。购货折扣情况详见例 4.6。

【参考代码】

```
#include <stdio.h>

void main()
{
    int c; //c 代表 100 的倍数
    float m, d, pay;
    printf("请输入购货款数:");
    scanf("%f", &m);
    if(m >= 2000)
        c = 20;
    else
        c = (int)m/100;
    switch(c)
    {
        case 0:
        case 1: d = 1.0; break;
        case 2:
        case 3:
        case 4: d = 0.95; break;
        case 5:
        case 6:
        case 7:
        case 8:
        case 9: d = 0.90; break;
        case 10:
        case 11:
        case 12:
        case 13:
        case 14:
        case 15:
        case 16:
        case 17:
        case 18:
        case 19: d = 0.85; break;
        case 20: d = 0.8; break;
    }
    pay = m * d;
    printf("应付款 = %7.2f 元, 实付款 = %7.2f 元\n", m, pay);
}
```

4.4 综合实例

【例 4.9】 采用 if 语句和 switch 语句编程,输入一个日期(年、月、日),判定这一日是一年的第多少天。

【解题思路】

要考虑闰年的 2 月份天数,假如 year 代表年份,判定闰年的条件当 $\text{year} \% 4 == 0 \&\& \text{year} \% 100 != 0$ 或者 $\text{year} \% 400 == 0$ 时 year 表示闰年。year 为闰年时 2 月份 29 天,year 不是闰年时 2 月份为 28 天。

【参考代码】

解法一

```
#include <stdio.h>
void main()
{
    int day, month, year, sum, day2, maxday;    //day2 表示 2 月最大天数, sum 表示第多少天
    printf("请输入日期(年,月,日):");
    scanf("%d - %d - %d", &year, &month, &day);
    if (year > 0 && month >= 1 && month <= 12 && day >= 1 && day <= 31)
    {
        if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0)    //year 是闰年
            day2 = 29;
        else
            day2 = 28;
        if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 || month == 10 || month == 12)
            maxday = 31;
        else if (month == 2)
        {
            if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0)    //year 是闰年
                maxday = 29;
            else
                maxday = 30;
        }
        if (day <= maxday)
        {
            if (month == 1) sum = day;
            else if (month == 2) sum = 31 + day;
            else if (month == 3) sum = 31 + day2 + day;
            else if (month == 4) sum = 31 + day2 + 31 + day;
            else if (month == 5) sum = 31 + day2 + 31 + 30 + day;
            else if (month == 6) sum = 31 + day2 + 31 + 30 + 31 + day;
            else if (month == 7) sum = 31 + day2 + 31 + 30 + 31 + 30 + day;
```

```

else if( month == 8) sum = 31 + day2 + 31 + 30 + 31 + 30 + 31 + day;
else if( month == 9) sum = 31 + day2 + 31 + 30 + 31 + 30 + 31 + 31 + day;
else if( month == 10) sum = 31 + day2 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + day;
else if( month == 11) sum = 31 + day2 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + day;
else if( month == 12) sum = 31 + day2 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30 + day;
printf( "%d - %d - %d 是 %d 年的第 %d 天\n", year, month, day, year, sum );
}
else
printf( "%d - %d - %d 是不合法日期\n", year, month, day );
}
else
printf( "%d - %d - %d 是不合法日期\n", year, month, day );
}

```

程序运行结果如图 4.12 所示。

解法二

```

#include <stdio.h>
void main()
{
    int day, month, year, sum, leap;
    printf( "\n 请输入日期(年,月,日):" );
    scanf( "%d - %d - %d", &year, &month, &day );
    switch( month ) /* 先计算某月以前月份的总天数 */
    {
        case 1: sum = 0; break;
        case 2: sum = 31; break;
        case 3: sum = 59; break;
        case 4: sum = 90; break;
        case 5: sum = 120; break;
        case 6: sum = 151; break;
        case 7: sum = 181; break;
        case 8: sum = 212; break;
        case 9: sum = 243; break;
        case 10: sum = 273; break;
        case 11: sum = 304; break;
        case 12: sum = 334; break;
        default: printf( "data error" ); break;
    }
    sum = sum + day; /* 再加上某天的天数 */
    if( year % 400 == 0 || ( year % 4 == 0 && year % 100 != 0 ) ) /* 判断是不是闰年 */
        leap = 1;
    else
        leap = 0;
    if( leap == 1 && month > 2 ) /* 如果是闰年且月份大于2,总天数应该加一天 */

```

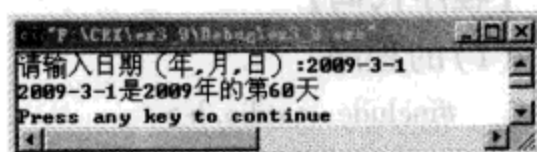


图 4.12 例 4.9 运行结果


```

sum = sum + 1;
printf("这天是当年的第 %d 天。\\n", sum);
}

```

【例 4.10】 有一函数：

$$y = \begin{cases} x & (-5 < x < 0) \\ x - 1 & (x = 0) \\ x + 1 & (0 < x < 10) \end{cases}$$

编写一程序,要求输入 x 的值,输出 y 的值。分别用:

- (1) 不嵌套的 if 语句;
- (2) 嵌套的 if 语句;
- (3) if-else 语句;
- (4) switch 语句。

【程序代码】

(1) 的程序:

```

#include <stdio.h>

void main()
{
    float x, y;
    printf("\\n 请输入 x: ");
    scanf("%f", &x);
    if(x <= -5 || x >= 10)
        printf("输入有误!");
    if((x > -5) && (x < 0))
    {
        y = x;
        printf("\\nx = %f, y = %f", x, y);
    }
    if(x == 0)
    {
        y = x - 1;
        printf("\\nx = %f, y = %f", x, y);
    }
    if((x > 0) && (x < 10))
    {
        y = x + 1;
        printf("\\nx = %f, y = %f", x, y);
    }
}

```

(2) 的程序:

```

#include <stdio.h>

void main()
{
    float x, y;
    printf("请输入 x: ");
    scanf("%f", &x);
    if(x > -5 && x < 10)
        if(x != 0)

```

```

if (x < 0) y = x;
else
    y = x + 1;
else
    y = x - 1;
else
    printf("输入有误\n!");
if (x > -5 && x < 10)
    printf("y = %f", y);
}

```

(3) 的程序:

```

#include <stdio.h>
void main()
{
    float x, y;
    printf("请输入 x:");
    scanf("%f", &x);
    if (x <= -5)
        printf("输入有误!");
    else if (x > -5 && x < 0)
        { y = x; printf("y = %f\n", y); }
    else if (x == 0)
        { y = x - 1; printf("y = %f\n", y); }
    else if (x > 0 && x < 10)
        { y = x + 1; printf("y = %f\n", y); }
    else
        printf("输入有误!");
}

```

(4) 的程序:

```

#include <stdio.h>
void main()
{
    float x, y;
    int s;
    printf("请输入 x:");
    scanf("%f", &x);
    if (x <= -5 || x >= 10)
        s = 0;
    if (x > -5 && x < 0)
        s = 1;
    if (x == 0)
        s = 2;
    if (x > 0 && x < 10)
        s = 3;
    switch(s)
    {
        case 0: printf("y = x\n"); break;
        case 1: printf("y = x + 1\n"); break;
        case 2: printf("y = x - 1\n"); break;
        case 3: printf("y = x + 1\n"); break;
        default: printf("输入有误!\n"); break;
    }
}

```

```

case 0: printf("y = x\n"); break;
case 1: printf("y = x + 1\n"); break;
case 2: printf("y = x - 1\n"); break;
case 3: printf("y = x + 1\n"); break;
default: printf("输入有误!\n"); break;
}

```

图 4.11 【例 4.11】
【程序参考】

```
#include <stdio.h>
```

```
void main()
```

```
{ int a = 2, b = 4, c = 3;
```

```
int x, y, z;
```

```
printf("a = %d, b = %d, c = %d\n", a, b, c);
```

```
printf("a < b < c\n", a < b < c);
```

```
z = 0; x = y = z;
```

```
z++; y++; x++;
```

```
printf("x = %d, y = %d, z = %d\n", x, y, z);
```

```
z = 0; x = y = z;
```

```
z++; y++; x++;
```

```
printf("x = %d, y = %d, z = %d\n", x, y, z);
```

```
z = 0; x = y = z;
```

```
z++; y++; x++;
```

```
printf("x = %d, y = %d, z = %d\n", x, y, z);
```

```
z = 0; x = y = z;
```

```
z++; y++; x++;
```

```
printf("x = %d, y = %d, z = %d\n", x, y, z);
```

图 4.13 程序运行结果示意图

图 4.13 程序运行结果示意图

图 4.13 程序运行结果示意图

图 4.13 程序运行结果示意图

```
<stdio.h>
```

```
void main()
```

```
{ int a = 2, b = 4, c = 3;
```

```
int x, y, z;
```

```
printf("a = %d, b = %d, c = %d\n", a, b, c);
```

```
printf("a < b < c\n", a < b < c);
```

```
z = 0; x = y = z;
```

```
z++; y++; x++;
```

```
printf("x = %d, y = %d, z = %d\n", x, y, z);
```

```
z = 0; x = y = z;
```

```
z++; y++; x++;
```

```
printf("x = %d, y = %d, z = %d\n", x, y, z);
```

```
z = 0; x = y = z;
```

```
z++; y++; x++;
```

```
printf("x = %d, y = %d, z = %d\n", x, y, z);
```

```
z = 0; x = y = z;
```

```
z++; y++; x++;
```

```
printf("x = %d, y = %d, z = %d\n", x, y, z);
```

```

{ case 0:printf("输入有误!\n");break;
  case 1:printf("y = %f\n",y = x);break;
  case 2:printf("y = %f\n",y = x - 1);break;
  case 3:printf("y = %f\n",y = x + 1);break;
}
}

```

【例 4.11】 运算符的优先级与逻辑运算符的使用。

【参考代码】

```

#include <stdio.h>
void main()
{ int a=5,b=4,c=3;
  int x,y,z;
  printf("%d,%d\n",!a,b);
  printf("%d\n",a>b>c);
  x=y=z=0;
  ++x||++y||++z;
  printf("x=%d,y=%d,z=%d\n",x,y,z);
  x=y=z=0;
  ++x&&++y||++z;
  printf("x=%d,y=%d,z=%d\n",x,y,z);
  x=y=z=0;
  ++x||++y&&++z;
  printf("x=%d,y=%d,z=%d\n",x,y,z);
  x=y=z=0;
  ++x&&++y&&++z;
  printf("x=%d,y=%d,z=%d\n",x,y,z);
}

```

程序运行结果如图 4.13 所示。

4.5 案例应用举例

【例 4.12】 利用 switch() 语句,编写“图书管理系统”菜单的选择。

【参考代码】

```

#include <stdio.h>
void main()
{ char ch1;
  printf("*****欢迎进入图书馆管理系统*****\n");
  printf(" *
  printf(" *          制作人:汤承林
  printf(" *      联系地址:江苏×××学院计算机科学与工程系
  printf(" * _____*_*_*_*_____

```

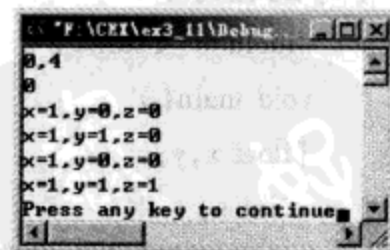


图 4.13 例 4.11 运行结果

```

printf(" * 2 (1) 请你选择操作类型: * \n");
printf(" * 1: <图书信息管理> * \n");
printf(" * 2: <借书卡管理> * \n");
printf(" * 3: <图书借还管理> * \n");
printf(" * 0: <退出> * \n");
printf(" ***** \n");
printf("请选择 0 -- 3:");
scanf("%c",&ch1);
getchar();
switch(ch1)
{ case '1':printf("欢迎进入图书管理模块!\n"); break;
  case '2':printf("欢迎进入借书卡管理模块!\n"); break;
  case '3':printf("欢迎进入借还书管理模块!\n"); break;
  case '0':exit(0);
  default:printf("无此操作,重新输入!\n"); break; }
}

```

程序运行结果如图 4.14 所示。

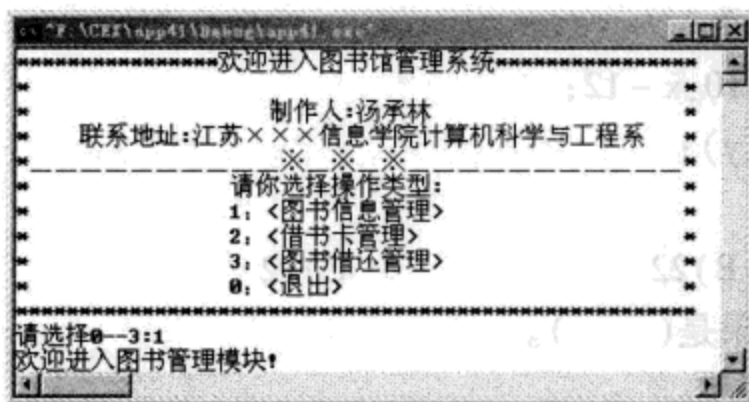


图 4.14 例 4.12 运行结果

本章小结

本章主要介绍了选择结构程序设计的基本思想和设计方法,详细介绍了关系表达式、逻辑表达式、条件表达式的书写规则;介绍了构成选择结构的各种条件语句。

if 语句主要用于单分支选择,if-else 语句主要用于双分支选择,if-else-if 语句和 switch 语句主要用于多分支选择。应当注意的是,这几种形式的选择语句一般来说是可以相互替代的。通过本章的学习,读者应该掌握选择结构程序设计的基本方法,并能用各种选择语句编写较复杂的程序。

习题四

一、选择题

1. 下列运算符中优先级最高的是()。

A) ! B) % C) - = D) &&)

2. 在 C 语言中, if 语句嵌套时, if 与 else 的配对关系是()。

A) 每个 else 总是与它上面最近的 if 配对 B) 每个 else 总是与最外层的 if 配对
C) 每个 else 与 if 配对关系是任意的 D) 每个 else 总是与它上面的 if 配对

3. 为表示关系 $x \geq y \geq z$, 应使用的 C 语言表达式是()。

A) $(x \geq y) \&\&(y \geq z)$ B) $(x \geq y) \text{ AND } (y \geq z)$
C) $(x \geq y \geq z)$ D) $(x \geq y) \&(y \geq z)$

4. 设 a、b 和 c 都是 int 型变量, 且 $a=5, b=6, c=7$, 则以下的表达式中, 值为 0 的表达式是()。

A) $a \&\&b$ B) $a \leq b$
C) $a || b + c \&\&b - c$ D) $!((a < b) \&\&!c || 1)$

5. 若运行时给变量 x 输入 12, 则以下程序的运行结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int x, y;
```

```
scanf("%d", &x);
```

```
y = x > 12 ? x + 10 : x - 12;
```

```
printf("%d\n", y);
```

```
}
```

A) 0 B) 22 C) 12 D) 10

6. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{ int w = 4, x = 3, y = 2, z = 1;
```

```
printf("%d\n", (w < x ? w : z < y ? z : x));
```

```
}
```

A) 1 B) 2 C) 3 D) 4

7. 若执行以下程序时从键盘上输入 3 和 4, 则输出结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int a, b, s;
```

```
scanf("%d %d", &a, &b);
```

```
s = a;
```

```
if (a < b) s = b;
```

```
s * = s;
```

```
printf("%d\n", s);
```

```
}
```

A)14 B)16 C)18

D)20

8. 下列程序运行结果是()。

```
#include <stdio.h>
void main()
{
    int i=0,j=0,a=6;
    if((++i>0)||(++j>0))
        a++;
    printf("%d,%d,%d\n",i,j,a);
}
```

A)0,0,6 B)1,0,7 C)1,0,6 D)1,1,7

9. 若 a 和 b 均是正整数型变量,以下正确的 switch 语句是()。

A) switch (pow(a,2) + pow(b,2)) (注:调用求幂的数学函数)

```
{ case 1: case 3: y = a + b; break;
  case 0: case 5: y = a - b;
}
```

B) switch (a * a + b * b);

```
{ case 3:
  case 1: y = a + b; break;
  case 0: y = b - a; break;
}
```

C) switch a

```
{ default : x = a + b;
  case 10 : y = a - b; break;
  case 11 : y = a * d; break;
}
```

D) switch(a + b)

```
{ case10: x = a + b; break;
  case11: y = a - b; break;
}
```

10. 运行以下程序后,输出()。

```
#include <stdio.h>
void main()
{
    int k = -3;
    if (k <= 0) printf(" * **\n");
    else printf("&&&\n");
}
```

A)####

B)&&&&

C) * * *

D) * * *

C)####&&&&

D)有语法错误不能通过编译

11. 有以下程序:

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int a = 1, b = 2, m = 0, n = 0, k;
```

```
k = (n = b > a) || (m = a < b);
```

```
printf("%d, %d\n", k, m);
```

```
}
```

程序运行后的输出结果是()。

A)0,0

B)0,1

C)1,0

D)1,1

12. 设 x, y, z, t 均为整型变量, 现有如下语句 $x = y = z = 1; t = ++x || ++y \&\& ++z;$, 则执行这个语句后 t 的值为()。

A)2

B)1

C)0

D)不定值

13. 若 $w = 1, x = 2, y = 3, z = 4$, 则条件表达式 $w < x ? w : y < z ? y : z$ 的值是()。

A)4

B)3

C)2

D)1

14. 请阅读以下程序:

```
#include <stdio.h>
```

```
void main()
```

```
{ int a = 5, b = 0, c = 0;
```

```
if (a = b + c) printf(" * * * \n");
```

```
else printf(" $ $ $ \n");
```

```
}
```

以上程序输出结果是()。

A)有语法错不能通过编译

B)可以通过编译但不能通过连接

C)输出 * * *

D)输出 \$ \$ \$

二、填空题

1. C 语言中用_____表示逻辑值“真”, 用_____表示逻辑值“假”。

2. C 语言中的逻辑运算符按优先级别是_____、_____、_____。

3. C 语言中逻辑运算符_____的优先级高于算术运算符。

4. 下面程序将两个数以从小到大的顺序输出, 试填充空格以完善程序。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
float a, b, _____;
```

```
scanf(_____, &a, &b);
```

```
if (a > b)
```

```
{
```

```
t = a;
```

```

    _____;
    b = t;
}
printf("%5.2f,%5.2f\n",a,b);
}

```

5. 请写出以下程序的输出结果_____。

```

#include <stdio.h>
void main()
{
    int a = 100;
    if (a > 100) printf("%d\n", a > 100);
    else printf("%d\n", a <= 100);
}

```

6. 下面程序是求分段函数的值,请填充空格以完善程序。

$$y = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

```

#include <stdio.h>
void main()
{

```

```

    int y;
    _____;
    scanf("%f",&x);
    if(_____) y = -1;
    else if(_____) y = 0;
    else y = 1;
    printf("x = %.2f y = %.2f\n",x,(float)y);
}

```

7. 以下程序输出的结果是_____。

```

#include <stdio.h>
void main()
{
    int a = 5, b = 4, c = 3, d;
    d = (a > b > c);
    printf("%d\n",d);
}

```

8. 以下程序运行后的输出结果是_____。

```

#include <stdio.h>

```

```

void main()

```



```

{
    int a=3,b=4,c=5,t=99;
    if(b<a&&a<c) t=a;a=c;c=t;
    if(a<c&&b<c) t=b;b=a;a=t;
    printf("%d%d%d\n",a,b,c);
}

```

9. 若变量已正确定义,以下语句段的输出结果是_____。

```

x=0,y=2,z=3;
switch(x)
{
    case 0: switch(y=2)
        {
            case 1: printf("*"); break;
            case 2: printf("%"); break;
        }
    case 1: switch(z)
        {
            case 1: printf("$");
            case 2: printf("*"); break;
            default: printf("#");
        }
}

```

10. 阅读下面语句,程序的执行结果是_____。

```

#include <stdio.h>
void main()
{
    int a=-1,b=1,k;
    if((++a<0)&&!(b--<=0))
        printf("%d,%d\n",a,b);
    else printf("%d,%d\n",b,a);
}

```

11. 阅读下面程序,执行后的输出结果是_____。

```

#include <stdio.h>
void main()
{
    int x,y,z;
    x=1;y=2;z=3;
    if(x>y)if(x>z)printf("%d",x);
    else printf("%d",y);
    printf("%d\n",z);
}

```

三、编程题

1. 某商场正开展打折促销活动。根据购买某种商品数量(x)多少给予不同的折扣。试编

写一程序。根据购买数量及该商品的单价,输出用户应付的金额。折扣信息如表 4.3 所示。

表 4.3 某商场的商品折扣信息表

数量	折扣情况
$x < 10$	无折扣
$10 \leq x < 20$	5% 折扣
$20 \leq x < 30$	10% 折扣
$30 \leq x < 40$	15% 折扣
$40 \leq x$	20% 折扣

编程采用:(1)if 语句;(2)二分支 if-else 嵌套语句;(3)多分支 if-else 嵌套语句;(4)switch 语句。

2. 有以下函数:

$$y = \begin{cases} x & (x < 1) \\ 2x - 1 & (1 \leq x < 10) \\ 3x - 11 & (x \geq 10) \end{cases}$$

编写一个程序,输入 x , 输出 y 的值。

2.1 例题

例 2.1 编写程序,计算下列分段函数的值。

解:分析如下。

该函数是一个分段函数,需要根据 x 的不同取值,选择不同的表达式进行计算。

根据题意,可以设计如下流程图。

流程图如下:开始 → 输入 x → 判断 $x < 1$? → 是,计算 $y = x$ → 输出 y → 结束; 否,判断 $1 \leq x < 10$? → 是,计算 $y = 2x - 1$ → 输出 y → 结束; 否,计算 $y = 3x - 11$ → 输出 y → 结束。

根据流程图,可以编写如下 C 语言程序。

程序代码如下:

【程序 2.1】

/* 计算分段函数的值 */

#include <stdio.h>

int

main()

{

double x;

double y;

scanf

%lf, &x);

if

(x < 1)

{

第 5 章 循环结构

在第 4 章我们通过选择和判断解决了多分支情况的问题,但有些问题仅仅通过选择和判断还不能解决,例如,要求输出九九乘法表。如果用前面所学的知识编制输出九九乘法表的一行的程序如下:

```
#include <stdio.h>
void main()
{
    printf("%d * %d = %2d %d * %d = %2d %d * %d = %2d ", 1, 1, 1 * 1, 1, 2, 1 * 2, 1, 3, 1 * 3);
    printf("%d * %d = %2d %d * %d = %2d %d * %d = %2d ", 1, 4, 1 * 4, 1, 5, 1 * 5, 1, 6, 1 * 6);
    printf("%d * %d = %2d %d * %d = %2d %d * %d = %2d\n", 1, 7, 1 * 7, 1, 8, 1 * 8, 1, 9, 1 * 9);
}
```

一行九九乘法表就要 3 条输出语句,如果九行全部输出,就要 27 条输出语句。假如要输出数 1 至 10 000,可能需要 10 000 条输出语句,这样既费时又费力。我们需要一种方法可以快速有效地执行重复性操作,在编程语言(如 C 语言)中可以通过循环结构来解决这类问题。

5.1 goto 语句

goto 语句称为无条件转向语句,它的一般形式为:

goto 语句标号;

goto 语句的作用是使程序的流程无条件转移到相应的语句标号处。它一般与 if 语句配合使用,构成循环。

语句标号是对语句的标识,应是合法的标识符,即由字母、数字和下画线组成,且第一字符必须是字母或下画线。注意不能用一个整数作为语句标号。

【例 5.1】求 1 到 100 之间的奇数和及偶数和。

【参考代码】

```
#include <stdio.h>
void main()
{
    int n = 1, sum1 = 0, sum2 = 0;
    loop: if (n <= 100)
    {
        if (n % 2 == 0)
            sum2 = sum2 + n;
        else
            sum1 = sum1 + n;
        n++;
        goto loop;
    }
}
```

```
printf("奇数和 = %d,偶数和 = %d\n",sum1,sum2);
```

程序执行结果如图 5.1 所示。

【注意点】

goto 语句是非结构化语句,大量使用会造成程序流向混乱,可读性差,因此结构化程序设计一般不用 goto 语句。

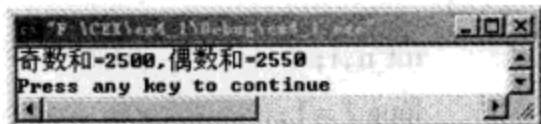


图 5.1 例 5.1 运行结果

5.2 for 语句

5.2.1 for 循环语句

1. for 循环语句的一般形式

```
for(表达式1;表达式2;表达式3)
```

```
{
    循环体;
}
```

其中,表达式1可以是赋值表达式、逗号表达式或函数调用表达式,通常是循环控制的初始化部分,为循环中所使用的变量赋初值;表达式2通常是关系表达式或逻辑表达式,是循环控制的条件,循环反复执行多次,必须在循环条件满足的情况下(即表达式2的值为非0)才能执行,否则循环终止;表达式3是赋值表达式或算术表达式,它使循环变量的值或循环控制条件得到修改,使循环只能进行有限次;循环体是循环结构中反复执行的语句,它可以是空语句(;)、单个语句或复合语句。

2. for 循环的执行过程

- (1) 计算表达式1的值;
- (2) 计算表达式2的值,若其值非0,则执行步骤(3);若为0,则转向步骤(6);
- (3) 执行循环体;
- (4) 计算表达式3的值;
- (5) 跳转到(2)继续执行;
- (6) 终止循环,执行 for 语句后的下一条语句。

for 语句的执行流程如图 5.2 所示。

for 循环语句是 C 语言中所提供的功能更强,使用更为广泛的一种循环语句。不仅用于循环次数已经确定的情况,而且可以用于循环次数不确定的且只给出循环结束条件的情况,它完全可以取代后面将要介绍的 while 循环和 do-while 循环。

【例 5.2】 输入一个整数 n,求 n!。

【解题思路】

由于 $n! = 1 * 2 * 3 * \dots * (n-1) * n$,定义一个变量 f 来表示积,并给出 f 的初始值为 1,设置一个控制循环次数变量 i,i 由 1 到 n,循环体为: $f = f * i$,实现连续多个数的积。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int n,i;
```

```
long f=1;
```

```
printf("请输入 n 的值:");
```

```
scanf("%d",&n);
```

```
for (i=1;i<=n;i++)
```

```
{
```

```
    f=f*i;
```

```
}
```

```
printf("n!=%ld\n",f);    //输出结果
```

```
}
```

//当循环体中仅有 1 个语句时，“{
|”可以省略

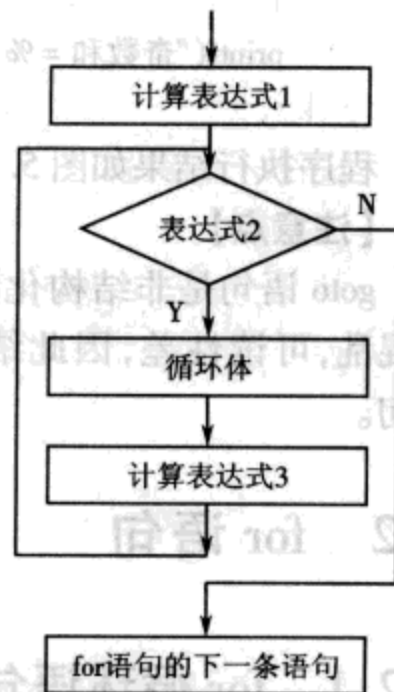


图 5.2 for 语句的执行流程图

程序运行结果如图 5.3 所示。

3. for 循环的格式说明

对例 5.2 进行 for 循环的格式说明。

(1) for 循环语句的一般形式中“表达式 1”可以省略,如例

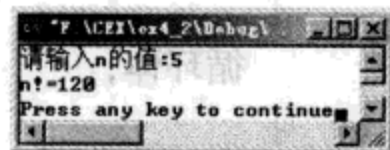


图 5.3 例 5.2 运行结果

5.2 中语句:

```
for(i=1;i<=n;i++) f=f*i;
```

可以改写为:

```
i=1;
```

```
for(;i<=n;i++) f=f*i;
```

(2) 如果“表达式 2”省略,即不判断循环条件,循环将无终止地进行下去。

```
for(i=1;;i++) f=f*i;
```

(3) “表达式 3”可以省略,但此时程序设计者必须保证循环能够通过其他设计正常结束。

```
for(i=1;i<=n;)
```

```
{ f=f*i;i++;}
```

(4) 可以只有“表达式 2”,无“表达式 1”和“表达式 3”。

```
i=1;
```

```
for(;i<=n;) { f=f*i;i++;}
```

(5) 省略“表达式 1”、“表达式 2”和“表达式 3”,必须要有能力终止循环。

```
i=1;
```

```
for(;;)
```

```
{
```

```
    f=f*i;
```

```
    i++;
```

```
    if(i>n)
```

```
        break;    //使用 break 跳出 for 循环
```

```
}
```


5.2.2 for 循环嵌套

一个循环体内可包含另外一个完整的循环结构。内嵌的循环中还可以继续嵌套循环,这就是循环的嵌套。

嵌套 for 循环语句一般形式为:

```
for( 表达式 11; 表达式 12; 表达式 13)
```

```
for( 表达式 21; 表达式 22; 表达式 23)
```

```
.....
```

```
for( 表达式 n1; 表达式 n2; 表达式 n3)
```

```
{ 循环体; }
```

注意:外层循环控制变量变化一次,内层循环控制变量要从初值到终值变化一轮。

【例 5.3】 输出九九表(三角形)。

【解题思路】

本题输出格式要求: $1 * 1 = 1$

$2 * 1 = 1 \quad 2 * 2 = 4$

$3 * 1 = 3 \quad 3 * 2 = 6 \quad 3 * 3 = 9$

.....

$9 * 1 = 9 \quad 9 * 2 = 18 \quad \dots \quad 9 * 9 = 81$

从格式要求看,用 i, j 两个变量分别控制行和列,第 i 行有积 i 个, j 从 1 变化到 i 。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
    int i, j;
```

```
    for (i = 1; i <= 9; i++)
```

```
    {
```

```
        for(j = 1; j <= i; j++)
```

```
            printf("%d * %d = %2d ", i, j, i * j);
```

```
            printf("\n");
```

```
    }
```

程序运行结果如图 5.4 所示。

【例 5.4】 编写一个程序,打印如图 5.5 所示菱形图案,并要求每边的字符数从键盘输入,如 4 或 5 等。

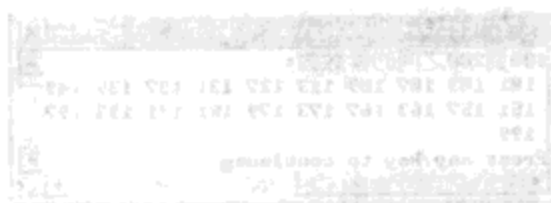
【解题思路】

从图案中看出,每行空格数和打印“*”的位置是有规律的,菱形的上半部分和下半部分可以分别打印,上半部分的规律为:第 i 行(i 从 1 到 n),首先打印 $n-i$ 个空格,然后考虑接下来从 1 到 $2*i-1$ 位置处打印的内容,只在边界打印“*”,在中间位置打印空格。后 $n-1$ 行的第 i 行(i 从 1 到 $n-1$),首先打印 i 个空格,然后考虑接下来从 1 到 $2*n-1-2*i$ 位置处打印的内容,只在边界打印“*”,在中间位置打印空格。本例需要使用嵌套循环,外层循环中要


```

printf(" ");
|
printf("\n");
|

```



程序运行的结果如图 5.6 所示。

【注意点】

(1) 例 5.4 中用 n 来控制循环的次数。

(2) 语句“if($k == 1 || k == 2 * i - 1$) printf(“*”); else printf(“ ”);”用来控制要打印的一行第一个“*”和最后一个“*”。

(3) 此例具有一定的代表性,当不打印一行中间的空格就是一个实心的菱形;当“*”换成数字或字符(每行不一样)时打印数字或字符菱形。



图 5.6 例 5.4 执行的结果

【例 5.5】 打印输出 100 到 200 之间的全部素数。

【解题思路】

素数是指只能被 1 和它本身整除的整数。算法比较简单,先将这个数被 2 除,如果整除,且该数又不等于 2,则该数不是素数;如果该数不能被 2 整除,再看是否被 3 整除,如果能被 3 整除,并且该数不等于 3,则该数不是素数,否则再判断是否被 4 整除,依次类推,该数只要是能被小于其本身的某个数整除时,就不是素数。

【参考代码】

```

#include <stdio.h>
void main()

```

```

{
    int i, j, n;

```

```

    n = 0;

```

```

    for(i = 100; i <= 200; i++)

```

```

    {
        for(j = 2; i % j != 0; j++)

```

```

        {
            if(i == j)

```

```

                printf("%4d", i);

```

```

                n++;

```

```

                if(n % 10 == 0)

```

```

                    printf("\n");

```

```

        }
    }
    printf("\n");
}

```

//n 用来控制每行输出素数的个数

//从 2 到 i 之间搜索第一个能被整除的数

//空语句,这个不能少,否则下面语句就是循环体了

//如果第一个能被整除的数等于该数本身,则说明该数为素数

//控制每行输出 10 个素数

程序运行的结果如图 5.7 所示。

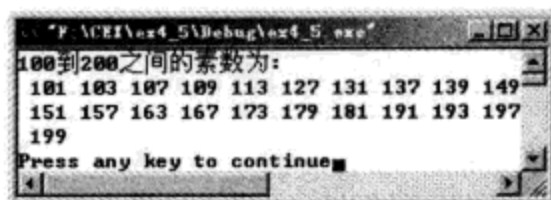


图 5.7 例 5.5 执行的结果

5.3 while 和 do-while 循环语句

5.3.1 while 循环语句

1. while 循环的一般形式

```
while (表达式)
```

```
{
    循环体;
}
```

2. 简要说明

“while”为循环关键字,表示该结构为 while 循环结构。圆括号中的表达式为循环控制条件,如果表达式结果为非 0 (“真”)值,则循环执行;如果表达式结果为 0 (“假”),则循环结束。控制循环的表达式为逻辑表达式,也可为其他复杂表达式,如赋值表达式、函数调用的结果等。无论是何种表达式,它的最终结果都是作为逻辑值。循环每执行一次,都要计算一次循环控制方式。

while 循环称为“当型”循环,“当型”循环的循环体可能在循环结束时一次都没执行过。

3. while 执行过程

循环开始,计算表达式,如果其值非 0 (逻辑“真”)则执行循环体,否则循环结束;当执行完循环体后,程序又回到循环开始处继续判断。while 循环的执行流程如图 5.8 所示。

【例 5.6】 输入整数 $n(n > 0)$, 求 $1 + 2 + 3 + \dots + n$ 的值。

【解题思路】

(1) 首先需要设置一个存放累加求和的变量 sum, 每一次加一个数到变量 sum 中;

(2) 再设置一个存放加数的变量 i, 每一次累加时被加到 sum 中且需要比前一个数大 1;

(3) 最后还需要设置一个结束循环的变量 n。

【参考代码】

```
#include <stdio.h>
void main()
{
    int i = 1, n, sum = 0;
```

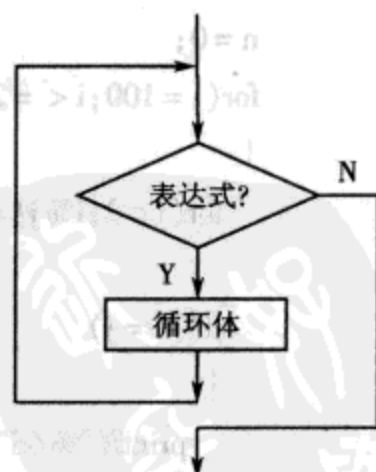


图 5.8 while 语句的执行流程图

```

printf("请输入 n 的值:");
scanf("%d",&n);
while(i <= n)    //累加的数 i 小于或等于终止数 n 就循环累加,否则结束循环
{
    sum = sum + i;
    i++;
}
printf("sum = %d\n",sum);

```

程序运行结果如图 5.9 所示。

【注意点】

(1) 存放累加求和的变量 `sum`, 定义后必须赋初值, 一般为 0, 否则由于定义的变量其原有初值不确定, 导致结果有误。

(2) `sum = sum + i;` 语句是一个累加求和功能的语句, 它是将变量 `sum` 中的数值加上变量 `i` 中的数值, 求和后再放入变量 `sum` 中, 此时变量 `sum` 中存放的数值已经变为新的数值, 原有数值已被覆盖掉。

【例 5.7】编写一个程序, 用于从控制台接收输入, 并将结果输出, 当按下 `^Z` (`Ctrl + Z`) 后, 程序结束。

【解题思路】

从控制台输入字符需要使用 `getchar()`, 存放在一个变量 `ch` 中, 用户每次输入一个字符相当于为 `ch` 重新赋值, 直到用户输入 `Ctrl + Z` 使循环条件变为假时, 才退出循环。`Ctrl + Z` 是 MS-DOS 操作系统中文件字符的末尾, 在程序中用 `EOF` 表示。

【参考代码】

```

#include <stdio.h>
void main()
{
    char ch;
    ch = getchar();
    while(ch != EOF)
    {
        putchar(ch);
        ch = getchar();
    }
}

```

程序运行结果如图 5.10 所示。

【注意点】

(1) 在图 5.10 中, 用户输入字符被反复显示。输入一串字符后, 只有按回车键, 这些字符才被送到内存缓冲区中, `putchar()` 语句将字符从内存缓冲区中取出送到显示器上。

(2) `while(ch != EOF) { putchar(ch); ch = getchar(); }` 可以用如下代码替换, 而且更简单。

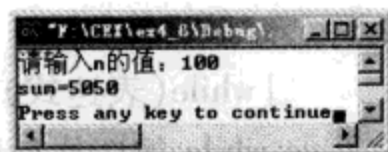


图 5.9 例 5.6 的运行结果

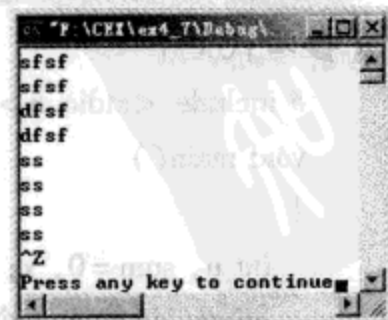


图 5.10 例 5.7 的运行结果


```
while((ch = getchar()) != EOF)
{
    putchar(ch);
}
```

5.3.2 do-while 循环语句

1. do-while 循环一般形式

```
do
```

```
{
```

```
    循环体;
```

```
} while(表达式);
```

2. do-while 循环执行过程

循环开始后,首先执行循环体,然后计算控制表达式。若其值非 0,回到循环开始,继续执行循环体,否则,循环结束。do-while 循环的执行过程如图 5.11 所示。

3. do-while 循环说明

do-while 循环与 while 循环相似,差别只在于两种循环的控制表达式所处位置不同。do-while 循环与 while 循环不同之处是:do-while 循环至少执行一次循环体,while 循环可能一次循环体都不会执行。

【例 5.8】编写一个程序,让用户输入一个正整数,然后计算各位数的和。例如,如果输入的数是 123,则计算结果是 6。

【解题思路】

假如输入的数存储在变量 n 中,用以下语句提取该数最右边一位的数字。

```
x = n % 10;
```

变量 x 存储该数最右边一位的数字的值。然后使用以下语句将取出的最右边的数累加到变量 sum 中。

```
sum += x;
```

再通过以下语句将 n 值去掉最右边的一位数,更新为一个新值:

```
n /= 10;
```

直到变量 n 等于变量 0 时,循环结束。

最终的和存储在变量 sum 中,并通过 `printf()` 语句显示出来。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int n, sum = 0, x;
```

```
    printf("请输入一个正数:");
```

```
    scanf("%d", &n);
```

```
    do
```

```
        /* do-while 循环开始 */
```

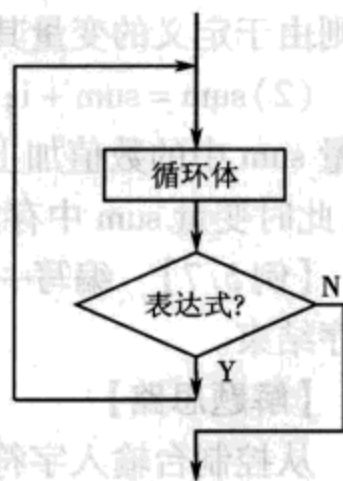


图 5.11 do-while 语句的执行流程图

```

{
    x = n % 10;
    sum += x;
    n /= 10;
}

/* sum = sum + x */
/* n = n / 10 */

while (n > 0); /* do-while 循环结束 */

printf("各位数字之和是:%d\n", sum);

```

程序的运行结果如图 5.12 所示。

5.3.3 for、while 和 do-while 循环嵌套

前面在 for 循环中介绍了循环嵌套概念,即一个循环体内又包含另一个完整的循环结构,称为循环嵌套。内嵌的循环中还可以包含循环嵌套,这就是多层循环。

三种循环(for 循环、while 循环和 do-while 循环)可以互相嵌套。例如,下列几种都是合法的形式:

```

(1) while( )
{
    :
    while( )
    {
        :
    }
}

```

```

(2) do
{
    :
    do
    {
        ...
    }
    while( );
} while( );

```

```

(3) for(;;)
{
    for(;;)
    ...
}

```

```

(4) while( )
{
    :
    do
    {
        ...
    }
    while( );
    :
}

```

```

(5) for(;;)
{
    :
    while( )
    {
        :
    }
}

```

```

(6) do
{
    :
    for(;;)
    {
        :
    }
    while( );
}

```

循环嵌套不能出现交叉嵌套。如图 5.13(a)是正确的,5.13(b)是不正确的。

【例 5.9】 编写一个程序,根据输入某个班级学生的成绩,计算该班级学生的平均成绩,班级的人数要求用户输入。根据输入的人数,分别录入学员的成绩。计算该班级学生的平均成绩,并显示计算结果。最后询问用户是否继续,直到用户自己确定要结束程序退出。



图 5.12 例 5.8 的运行结果

【参考代码】

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int class_id, stud_num, i, score, score_sum;
    char choice;
    do
    {
        choice = 'Y';
        printf("请输入班级号:");
        scanf("%d", &class_id);
        score_sum = 0;
        if(class_id > 0)
        {
            printf("请输入班级的学生总数:");
            scanf("%d", &stud_num);
            i = 1;
            while(i <= stud_num)
            {
                printf("输入学号 %d 的成绩:", i);
                scanf("%d", &score);
                score_sum = score_sum + score;
                i++;
            }
            /* while 循环结束 */

            printf("该班级的学生的平均成绩为: %d\n", score_sum/stud_num);
            printf("是否输入另一个班级的分数 (Y/N)? \n");
            choice = getchar();
            /* 用户选择是否继续 */

            while(choice == 'Y' || choice == 'y'); /* do-While 循环结束 */
        }
    }
}

```

程序运行结果如图 5.14 所示。

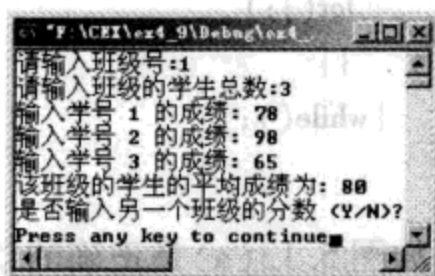


图 5.14 例 5.9 的运行结果

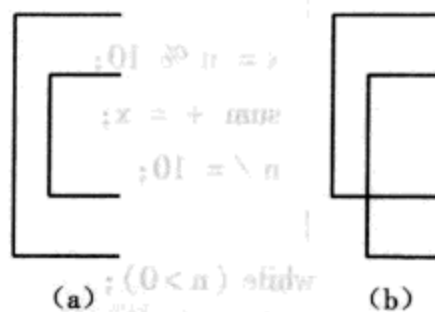


图 5.13 循环嵌套

5.4 break 语句和 continue 语句

在循环执行过程中,我们有时不知道循环要执行的次数,所以需要找到一种机制,在满足某种条件情况下跳出循环。这时,需要根据条件跳出循环的一些语句,终止循环的条件可在循环体内。C 语言提供了 break 和 continue 两个关键字,用于改变程序的控制流。

5.4.1 break 语句

break 语句通常用在 switch 语句及循环语句中,当 break 用于 switch 语句中时,可使程序跳出 switch 语句而执行 switch 语句后面的语句;当 break 语句用于三种循环(for 循环、while 循环和 do-while 循环)中时,可使循环终止而执行循环后面的语句。

break 语句实际上就是为了使能方便地描述从循环执行中退出的动作。通常应把 break 语句放在条件语句控制之下,以便在某条件满足时立即结束循环。

【例 5.10】 统计从键盘输入的若干个字符中有效字符的个数,以换行符作为输入结束。有效字符是指每一个空格前面的字符,若输入的字符中没有空格,则有效字符为除了换行符之外的所有字符。

【解题思路】

用一个 ch 来保存 getchar() 输入的字符,用 n 表示输入有效字符的个数,以“\n”结束字符的输入,但在输入换行符之前如果有空格符则应结束统计有效字符个数。

【参考代码】

```
#include <stdio.h>
void main()
{
    int n=0; //n=0 必须要有,否则出错
    char ch;
    printf("请输入一行字符:");
    while((ch = getchar()) != '\n') //'\n'为换行符
    {
        if(ch == ' ') //跳出循环
            break;
        else //计数
            n++;
    }
    printf("有效字符个数:%d\n",n);
}
```

程序运行结果如图 5.15 所示。

【注意点】

(1) break 语句通常在循环中与条件语句一起使用。若条件满足,将跳出循环,控制流转向循环后面的语句。

(2) 如果已执行 break 语句,就不会执行循环体中位于 break

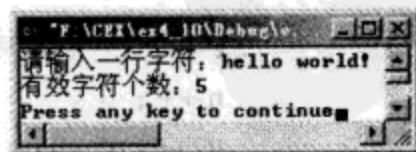


图 5.15 例 5.10 的运行结果

语句后的语句。

(3) 在多层循环中, 一个 break 语句只能向外跳出一层循环。

5.4.2 continue 语句

continue 语句只能用在循环里。continue 语句的作用是跳过循环体中剩余的语句而准备执行下一次循环。对于 while 和 do-while 循环, continue 执行之后的动作是条件判断; 对于 for 循环, 随后的动作是变量更新。

注意 break 语句和 continue 语句的差别。break 语句导致循环终止, 使程序控制流转向这个循环语句之后; continue 引起的则是循环内部的一次控制转移, 使执行控制跳到循环体的最后, 相当于跳过循环体里这个语句后面的那些语句, 继续下一次循环。如图 5.16 说明了 break 语句和 continue 语句引起的控制转移情况。

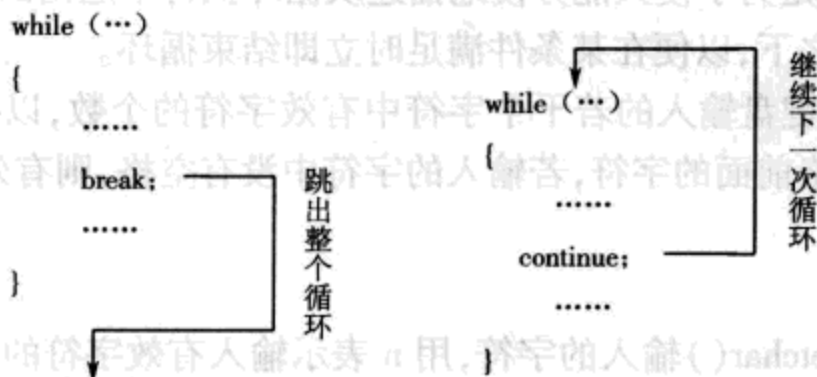


图 5.16 break 和 continue 语句引起的控制转移

【例 5.11】 输出 200 以内能被 5 整除的正整数, 每行要求输出 10 个数。

【解题思路】

(1) 用一个 n 来计算输出的数据个数, 当 $n \% 10$ 等于 0 时用换行语句: `printf("\n")`。

(2) 用 i 来表示被验证的数, 当 $i \% 5$ 等于 0 输出 i 的值, 并 $n++$, 否则使 $i++$, 跳到下一次循环。

【参考代码】

```
#include <stdio.h>
void main()
{
    int n=0,i; //n=0 用来控制每行输出数字个数
    printf("1--200 之间能被 5 整除的数:\n");
    for(i=1;i<=200;i++)
    {
        if (i%5==0)
        {
            printf("%4d",i); //输出能被 5 整除的数
            n++;
            if(n%10==0)
            {
                printf("\n"); //控制一行输出 10 个数
            }
        }
    }
}
```



```

else
    continue;
    //跳到下一次循环
}
printf("\n");

```

程序运行结果如图 5.17 所示。

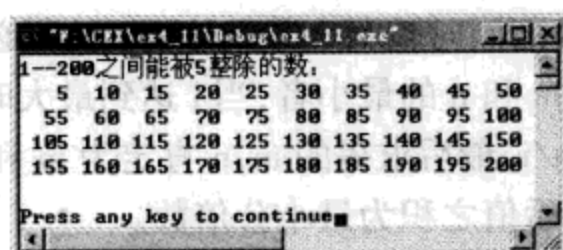


图 5.17 例 5.11 的运行结果

5.5 综合实例

【例 5.12】 输入一串字符,计算输入的字母个数,空格个数,数字字符个数,其他字符个数。

【解题思路】

字母指的是 26 个大小写字母,其他字符指非 26 个大小写字母、空格和数字字符。

【参考代码】

```

#include <stdio.h>

main()
{
    int i=0,j=0,k=0,m=0; /* i 字母个数,j 空格个数,k 数字个数,m 其他字符个数 */
    char c;
    printf("\n 请输入一串字符:");
    while ((c = getchar()) != '\n')
    {
        if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')
            i++;
        else if (c == ' ')
            j++;
        else if (c >= '0' && c <= '9')
            k++;
        else
            m++;
    }
    printf("i = %d,j = %d,k = %d,m = %d\n",i,j,k,m);
}

```

程序运行结果如图 5.18 所示。

【例 5.13】输入两个正整数 m 和 n , 求它们的最大公约数和最小公倍数。

【解题思路】

(1) 用 m, n 表示输入的两个数, 把小的数放在 m 中, 大的数放在 n 中。

(2) 用一个 i 作为循环控制变量, 并被 m, n 除, 如果同时被整除, 则 i 必为两数之因数, 能同时被两个数整除的 i 的积必为最大公约数。

(3) i 的最大值为变化后的 m 和 n 的最小者, 当 i 达到最大时循环结束。

(4) 反复实施 $m = m/i, n = n/i$, 最后求得的 m, n 肯定为 m 和 n 的最小公倍数的因数。

(5) i 的累积值与 m, n 的最后值之积为最小公倍数。

【参考代码】

```
#include <stdio.h>

void main()
{
    int m, n, max = 1, min = 1, i, temp; //max 存放最大公约数, min 存放最小公倍数
    printf("请输入两个正整数:"); //提示输入行
    scanf("%d %d", &m, &n);
    if (m > n) //交换两个数, m 中存放小的数, n 中存放大的数
    {
        temp = m;
        m = n;
        n = temp;
    }
    i = 2;
    while(i <= m)
    {
        if(m%i == 0 && n%i == 0) //如果 m, n 同时整除 i, 则 i 为 m 和 n 的因数
        {
            m = m/i;
            n = n/i;
            max = max * i; //累积同时为 m 和 n 的因数, 最后求得者为最大公约数
        }
        i++;
    }
    min = max * m * n; //求得最小公倍数
    printf("最大公约数: %d, 最小公倍数: %d\n", max, min);
}
```

程序运行结果如图 5.19 所示。

【注意点】

在众多的 C 语言教材中, 大多采用“辗转相除法”的算法实现求最大公约数和最小公倍数, 但该方法高职高专学生理解起来较难, 这里就不再讨论了。

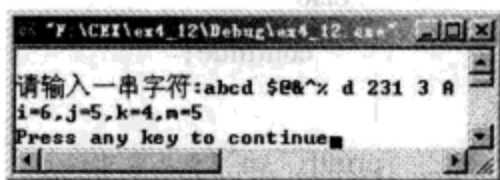


图 5.18 例 5.12 的运行结果

【例 5.14】编写程序输出斐波拉契(Fibonacci)数列中的前 20 个数。斐波拉契数列的组成是:1, 1, 2, 3, 5, 8, 13, 21, 34, ...

【解题思路】

通过对该数列的观察发现:数列的第 3 个数为第 1 个数与第 2 个数的和;第 4 个数为第 2 个数与第 3 个数的和;依次类推,第 i 个数是第 $i-1$ 个数与第 $i-2$ 个数之和($i \geq 3$)。

假定 f_i 表示第 i 个数,则 $f_1 = 1, f_2 = 1, f_3 = f_1 + f_2, \dots, f_i = f_{i-1} + f_{i-2}$ 。

【参考代码】

```
#include <stdio.h>

void main()
{
    int f1, f2, f3, i;
    f1 = 1; f2 = 1;
    printf("前 20 个斐波拉契数:\n");
    printf("%6d%6d", f1, f2);
    for(i = 3; i <= 20; i++)
    {
        f3 = f1 + f2;
        printf((i % 5 == 0) ? "%6d\n" : "%6d", f3);
        f1 = f2;
        f2 = f3;
    }
}
```

程序运行的结果如图 5.20 所示。

【注意点】

语句“`printf((i % 5 == 0) ? "%6d\n" : "%6d", f3);`”中使用“?:”运算符,它实现打印输出格式的控制,表示每输出 5 个斐波拉契数后换一行。

该语句等价于如下 if-else 语句:

```
if(i % 5 == 0)
    printf("%6d\n", f3);
else
    printf("%6d", f3);
```

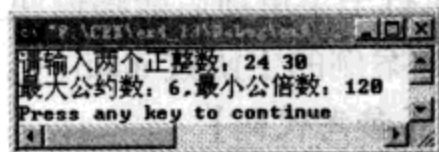


图 5.19 例 5.13 的运行结果

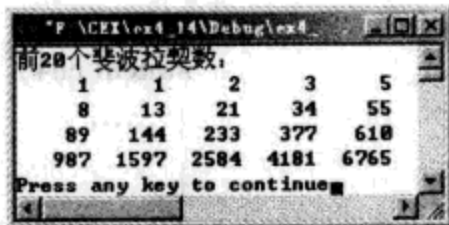


图 5.20 例 5.14 的运行结果

5.6 案例应用举例

【例 5.15】利用 do-while 循环实现“图书管理系统”主菜单与子菜单操作。

【解题思路】

使用 do-while 循环结合 switch 语句,对主菜单项进行选择时,可以进入子菜单项;在选择

并执行子菜单项后能够返回上一级的菜单,菜单项的选择能够反复进行。详细参考代码见第 12 章案例的主函数 main()。

【参考代码】

```
void main()
{
    char ch1, ch2, ch3, ch4;
    do
    {
        ..... //详见第 12 章案例
        switch( ch1)
        {
            case '1':
            do
            {
                ..... //详见第 12 章案例
                switch( ch2)
                {
                    ..... //详见第 12 章案例
                }
            } while( ch2 != '0'); break;
            case '2':
            do
            {
                ..... //详见第 12 章案例
                switch( ch3)
                {
                    ..... //详见第 12 章案例
                }
            } while( ch3 != '0'); break;
            case '3':
            do
            {
                ..... //详见第 12 章案例
                switch( ch4)
                {
                    ..... //详见第 12 章案例
                }
            } while( ch4 != '0'); break;
            case '0': exit(0); //exit(0)表示正常退出并返回操作系统
        }
    } while( ch1 != '0');
}
```

本章小结

(1) 本章介绍了循环结构程序设计的方法,主要介绍了 3 种循环语句:while、do-while 和 for 语句。

(2) 使用 while、do-while 循环语句时,循环控制变量的初值在 while、do-while 语句之前赋值,对循环控制变量的修改则是在循环体内完成;而 for 通常是在表达式 1 的位置实现对循环控制变量的初始化,在表达式 3 的位置对循环控制变量进行修改。

(3)另外,还介绍了 goto、break 和 continue 语句的用法。但尽量不用 goto 语句,因为它会影响程序的可读性。

习题五

一、选择题

1. 以下程序段的输出结果是()。

```
int k,j,s;
for (k=2;k<6;k+=2)
{
    s=1;
    for (j=k;j<6;j++)
        s+=j;
    printf("%d\n",s);
}
```

A) 9 B) 1

C) 11

D) 10

2. void main()

```
{
    int i,s=1;
    for (i=1;i<50;i++)
        if (!(i%5)&&!(i%3))s+=i;
    printf("%d\n",s);
}
```

程序的输出结果是()。

A) 409 B) 277

C) 1

D) 91

3. 以下程序段的输出结果是()。

```
int i,j,m=0;
for (i=1;i<=5;i+=4)
    for (j=3;j<=19;j+=4)
        m++;
printf("%d\n",m);
```

A) 10 B) 15

C) 20

D) 25

4. 已知 int t=0;

```
while (t=1)
{ ... }
```

则以下叙述正确的是()。

A) 循环控制表达式的值为 0

C) 循环控制表达式不合法

B) 循环控制表达式的值为 1

D) 以上说法都不对

5. 以下程序段的输出结果是()。


```
int n = 10;
while (n > 7)
{
    n--;
    printf("%d\n", n);
}
```

A) 10 9 8 B) 9

8

7

C) 10

9

8

6. 下列程序的输出结果是()。

```
#include <stdio.h>
void main()
{
    int i, a = 0, b = 0;
    for(i = 1; i < 10; i++)
    {
        if(i % 2 == 0)
        {
            a++;
            continue;
        }
        b++;
    }
    printf("a = %d, b = %d", a, b);
}
```

A) a = 4, b = 4

B) a = 4, b = 5

C) a = 5, b = 4 D) a = 5, b = 5

7. 以下程序段的输出结果是()。

```
int x = 3;
do
{
    printf("%3d", x--);
}
while (!(--x));
```

A) 1

B) 3 0

C) 1 -2 D) 死循环

8. 设有以下程序段:

```
int x = 0, s = 0;
while(!x != 0) s += ++x;
printf("%d", s);
```

则()。

A) 运行程序段后输出 0

C) 程序段中的控制表达式是非法的

B) 运行程序段后输出 1

D) 程序段执行无限次

9. 以下程序的输出结果是()。

```
#include <stdio.h>
void main()
{
```

```

int i, sum;
for (i = 1; i < 6; i++)
    sum += i;
printf("%d\n", sum);
}

```

A) 15 B) 14

10. 阅读如下程序段

```

#include <stdio.h>
void main()
{
    int x, a, b;
    scanf("%d", &x);
    a = b = x;
    for(; x != 0;)
    {
        if(x < b) b = x;
        if(x > a) a = x;
        scanf("%d", &x);
    }
    printf("a = %d, b = %d\n", a, b);
}

```

现输入如下: (" " 表示空格)

34]56]23]45]5]56]7]12]365]49]48]57]87]7]6]7569]789]0 <Enter>

输出结果是()。

A) a = 7569, b = 789 B) a = 5, b = 365

C) a = 7, b = 789 D) a = 7569, b = 5

11. 以下程序的输出结果是()。

```

#include <stdio.h>
void main()
{
    int y = 10;
    for (; y > 0; y--)
        if (y % 3 == 0)
        {
            printf("%d", --y);
            continue;
        }
}

```

A) 741

B) 852

C) 963

D) 875421

12. 以下程序的运行结果为()。

```

#include <stdio.h>

```

```
void main()
```

```
{
```

```
    int m,n;
```

```
    for(m=0,n=10;m<n;m+=3,n--);
```

```
    printf("%d,%d\n",m,n);}
```

A) 6,7

B) 7,6

C) 9,7

D) 7,9

13. 若 x 是 int 型变量,以下程序段的输出结果是()。

```
for(x=3;x<6;x++)
```

```
printf((x%2)? (" * * %d") : "##%d\n"),x);
```

A) **3

B) ##3

C) ##3

D) **3##4

##4

**4

**4##5

**5

**5

##5

14. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int a,i;
```

```
    a=0;
```

```
    for(i=1;i<5;i++)
```

```
    {
```

```
        switch(i)
```

```
        case 0:
```

```
        case 3: a += 2;
```

```
        case 1:
```

```
        case 2: a += 3;
```

```
        default: a += 5;
```

```
    }
```

```
}
```

```
printf("%d\n",a);
```

```
}
```

A) 31

B) 13

C) 10

D) 20

15. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int i;
```

```
    for(i=1;i<=5;i++)
```

```
    {
```

```

    if (i%2) printf("*");
    else continue;
    printf("#");
}
printf("$ \n");
}

```

- A) #####\$ B) #####* \$ C) ###\$ () D) #####\$

16. 有以下程序:

```

#include <stdio.h>
void main()
{
    int c;
    while( (c = getchar()) != '\n')
    {
        switch(c - '2')
        {
            case 0:
            case 1: putchar(c + 4);
            case 2: putchar(c + 4); break;
            case 3: putchar(c + 3);
            case 4: putchar(c + 3); break;
        }
    }
    printf("\n")
}

```

从第一列开始输入以下数据, <CR> 代表一个回车符。

2743 <CR>

程序的输出结果是()。

- A) 66877 B) 668966 C) 6677877 D) 6688766

17. 以下叙述正确的是()。

- A) do-while 语句构成的循环不能用其他语句构成的循环代替
 B) do-while 语句构成的循环只能用 break 语句退出
 C) 用 do-while 语句构成循环时, 只有在 while 后的表达式为非零时结束循环
 D) 用 do-while 语句构成循环时, 只有在 while 后的表达式为零时结束循环

18. 有如下程序:

```

#include <stdio.h>
void main()
{
    int n = 9;

```

```

while(n > 6)
{
    n--;
    printf("%d", n);
}

```

该程序的输出结果是()。

- A) 987 B) 876 C) 8765 D) 9876

19. 以下程序的输出结果是()。

```

#include <stdio.h>
void main()
{
    int x, i;
    for (i = 1; i <= 100; i++)
    {
        x = i;
        if (++x % 2 == 0)
            if (++x % 3 == 0)
                if (++x % 7 == 0)
                    printf("%d ", x);
    }
    printf("\n");
}

```

- A) 39 81 B) 42 84 C) 26 68 D) 28 70

20. 有以下程序:

```

#include <stdio.h>
void main()
{
    int x = 0, y = 0, i;
    for (i = 1; ; ++i)
    {
        if (i % 2 == 0) { x++; continue; }
        if (i % 5 == 0) { y++; break; }
        printf("%d, %d", x, y);
    }
}

```

程序的输出结果是()。

- A) 2, 1 B) 2, 2 C) 2, 5 D) 5, 2

21. 在下列选项中, 没有构成死循环的是()。

- A) `int i = 100;`
`while(1)`
`{ i = i % 1; } 0 + 1;`
`if(i > 100) break;`
`}`

B) for(;;);

C) int k = 10000;

do { k++; } while (k > 10000);

D) int s = 36;

while (s) --s;

二、填空题

1. 当执行以下程序段后, i 的值是_____, j 的值是_____, k 的值是_____。

```
int a, b, c, d, i, j, k;
```

```
a = 10; b = c = d = 5; i = j = k = 0;
```

```
for (; a > b; ++b)
```

```
    i++;
```

```
while (a > ++c)
```

```
    j++;
```

```
do
```

```
{
```

```
    k++;
```

```
} while (a > d++);
```

```
printf("%d, %d, %d\n", i, j, k);
```

2. 以下程序的运行结果是_____。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int i = 1;
```

```
    while(i <= 15)
```

```
    if( ++i % 3 != 2)
```

```
        continue;
```

```
    else
```

```
        printf("%d", i);
```

```
        printf("\n");
```

```
}
```

3. 以下程序的输出结果是_____。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int k, n, m;
```

```
    n = 10; m = 1; k = 1;
```

```
    while (k <= n)
```

```
{
```

```
        m * = 2;
```

```

        k++;
    }
    printf("%d\n", m);
}

```

4. 以下程序运行结果是_____。

```

#include <stdio.h>
void main()
{
    int n=4, i, j;
    long s=0, s1;
    for(i=1; i<=n; i++)
    {
        s1=1;
        for(j=1; j<=i; j++)
            s1=s1*j;
        s=s+s1;
    }
    printf("%ld\n", s);
}

```

5. 以下程序段的输出结果是_____。

```

#include <stdio.h>
void main()
{
    int x=2;
    while (x--);
    printf("%d\n", x);
}

```

6. 若所用变量都已正确定义, 请问下列 for 循环语句中 printf() 执行的次数是_____。

```

a=2, b=10;
for(i=b; i<=a; i--, a++)
    printf("%3d", i);

```

7. 以下程序的输出结果是_____。

```

#include <stdio.h>
void main()
{
    int i=0, sum=1;
    do
    {
        sum+=i++;
    }
    while(i<10);
    printf("%d", sum);
}

```

```

    } while (i < 5);
    printf("%d\n", sum);
}

```

8. 用下列程序计算数的阶乘, 请填充空格以完善程序。

```

#include <stdio.h>
main()
{
    int i, n;
    long np;
    scanf("%d", &n);
    np = ____;
    for(i = 2; i <= n; i++)
        np = ____;
    printf("n = %d, %d! = %ld\n", n, n, np);
}

```

9. 有以下程序段:

```

s = 1.0;
for(k = 1; k <= n; k++) s = s + 1.0 / (k * (k + 1));
printf("%f\n", s);

```

请填空, 使下面程序段的功能与上面程序段完全等同。

```

s = 0.0;
____;
k = 0;
do
{
    s = s + d;
    ____;
    d = 1.0 / (k * (k + 1));
}
while (____);
printf("%f\n", s);

```

10. 下列程序是计算圆周率(π)的近似值($\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$), 请填充空格以完善该程序。

```

#include <stdio.h>
#include <math.h>
void main()
{
    int s;
    float n, t, ____;

```

```

t = 1; pi = 0;
n = 1.0; s = 1;
while(____ > 1e-6)
{
    pi = ____;
    n = n + 2;
    s = -s;
    t = s/n;
}
pi = pi * 4;
printf("pi = %10.6f\n", pi);
}

```

11. 以下程序的功能是从键盘上输入若干学生的成绩,统计并输出最高成绩和最低成绩,当输入非正数时结束输入。

```

main()
{ float x,amax,amin;
  scanf("%f",&x);
  amax = x; amin = x;
  while (____)
  { if (x > amax) amax = x;
    if (____) amin = x;
    scanf("%f",&x);
  }
  printf("\namax = %f\namin = %f\n",amax,amin);
}

```

12. 下列程序的输出结果是_____。

```

#include <stdio.h>
void main()
{ int i;
  for(i = 1; i + 1; i++)
  { if(i > 4)
    { printf("%d\n", i);
      break; }
    printf("%d\n", i++); } }

```

三、编程题

1. 编写一程序,计算 $1! + 2! + \dots + n!$ 的值,其中 n 的值由用户输入。
2. 编写一个程序输出如下图案:

A
BBB
CCCCC

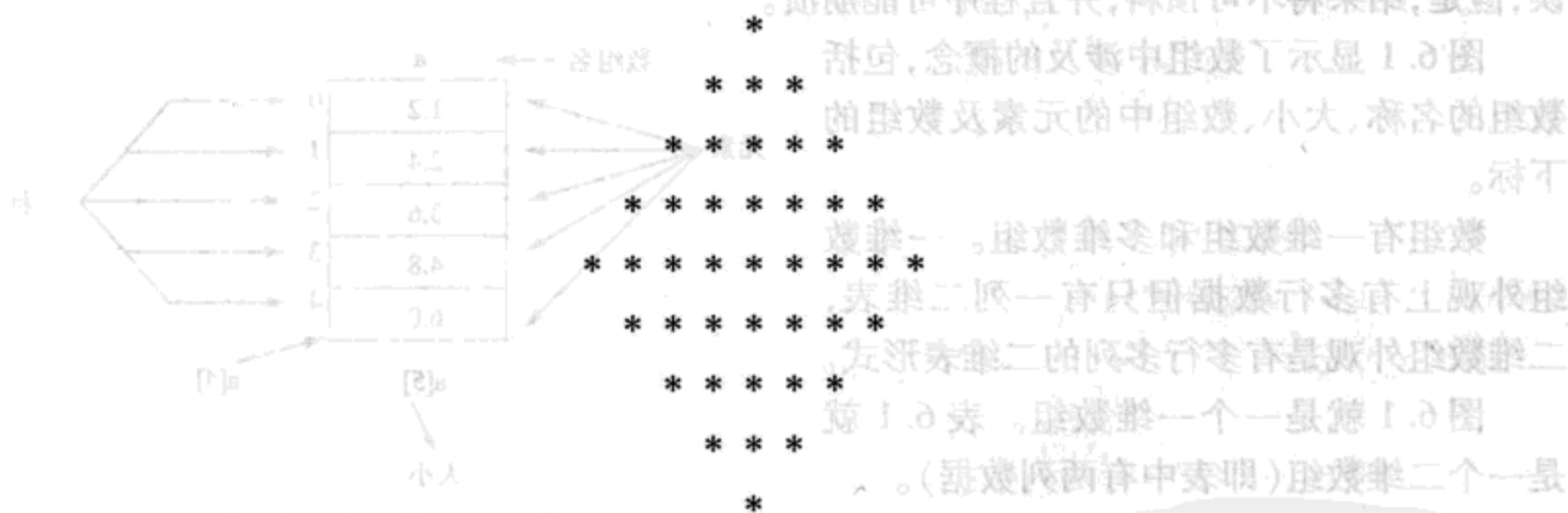
DDDDDDDD
EEEEEEEEEE

3. 以表格形式显示 1 到 10 的乘法,如图 5.21 所示。

X \ Y	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

图 5.21 乘法表

4. 编写程序,打印以下图形。



5. 编写一个程序,输入某个班级的人数;根据输入的人数,分别录入学生的成绩;计算该班级学生的平均成绩,并显示计算结果;最后询问用户是否继续,直到用户自己确定要结束程序退出。

85	87	89
87	87	89
89	88	89

图 5.22 打印图形

第 6 章 数组

在前几章中,使用的都是基本数据类型(整型、字符型、实型)的数据,此外,C 语言还提供了数组类型、结构体类型、共用体类型等构造类型的数据。它们是由基本数据按一定规则组成的。

在程序运行过程中,使用变量可以存储单个数据,而要存储多个相关的数据则要使用数组。数组是程序设计中处理较复杂问题必须使用的数据存储方式。

数组是在内存中可以连续存储多个元素的结构。数组中的所有元素必须属于同一数据类型,不能将数据类型不同的数据存储在同一个数组中。不管数组中含有多少个元素,该数组都只有一个名称。数组元素在数组里顺序排列编号,首元素的编号规定为 0,其他元素依次递增编号。数组元素的编号称为元素的下标,数组中的每个元素都可以通过下标访问。由于元素是按顺序存储的,因此,可以通过下标快速地访问到每个元素,数组大小是数组可容纳的元素的最大数量。在数组的处理过程中,如果数组的下标值超过此大小,C 语言不会导致语法错误,但是,结果将不可预料,并且程序可能崩溃。

图 6.1 显示了数组中涉及的概念,包括数组的名称、大小、数组中的元素及数组的下标。

数组有一维数组和多维数组。一维数组外观上有多行数据但只有一列二维表,二维数组外观是有多行多列的二维表形式。

图 6.1 就是一个一维数组。表 6.1 就是一个二维数组(即表中有两列数据)。

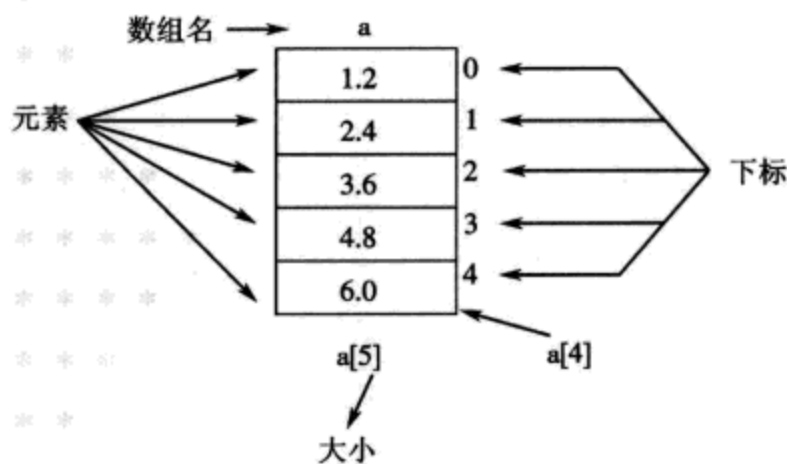


图 6.1 数组相关概念

表 6.1 一个二维数组

姓名	理论成绩	实验成绩
张三	78	85
李四	76	78
王五	56	98

6.1 一维数组

仅含有一个下标的数组称为一维数组,其中的数据排列成一行。C 语言规定,一维数组的下标是从 0 开始标记的。

6.1.1 一维数组的定义及初始化

1. 一维数组的定义

数组的类型实际上是指数组元素的取值类型。对于同一个数组,其所有的数据类型都是相同的。数组名的书写规则应符合标识符的规定。数组名不能与其他变量名相同。

在C语言中,与变量的定义一样,数组也遵循“先定义后使用”的原则。一维数组的一般形式为:

类型说明符 数组名[常量表达式];

例如,一个含有30个元素的整型数组定义为:

```
int a[30];
```

a[30]数组共有30个元素,在内存中,这30个数组元素共占用30个连续的存储单元,每个存储单元中只能存储一个整数,第一个元素对应的存储单元的地址称为数组首地址。(在Visual C++6.0中每个int单元占用4个字节)

在定义一维数组时,应注意以下几点。

(1)类型说明符用来说明数组元素属于何种数据类型,类型说明符可以是C语言中任何有效数据类型,包括结构体数据类型,如int、float、char、指针和结构体等。

(2)数组名是用户自定义的,用于引用该数组。

(3)常量表达式表示数组元素个数,也称为数组的长度,它的结果必须是一个正整数值或正整数常量。但要注意数组在内存中所占的字节数=数组长度×元素类型长度。图6.1中数组a[5]占用的内存空间字节数=5×4=20字节(Byte)。

(4)数组名后的方括号“[]”不能少,也不可用其他括号。如用a(5)(BASIC语言使用)表示数组是错误的。

(5)常量表达式中一般包括整型常量、字符常量和符号常量,但不能包括实型(符号)常量或字符串(符号)常量。例如:

```
#define N 2
```

```
.....
```

```
int a[10+'1'+N];
```

是正确的。其中10是整型常量,'1'是字符常量,相当于49('1'的ASCII码值),N是符号常量。实际使用时除非特殊需要,一般都只要用一个整型常量值来定义数组的长度。

(6)常量表达式中不能包括变量。

(7)数组下标是从0开始,所以a[5]的数组元素应为a[0]、a[1]、a[2]、a[3]、a[4],它的最大下标是4而非5。

(8)相同类型的数组、变量可以放在一起定义。例如:

```
int a[5],i,j,x[10];
```

定义了两个整型数组a[5]、x[10]和两个整型变量i、j。

2. 一维数组的初始化

数组的初始化是指在定义数组的同时为数组元素赋初始值。一维数组初始化的一般形式为:

类型说明符 数组名[常量表达式]={值1,值2,...,值n};

其中“{ }”括号中的各个值依次对应赋给数组中的各个元素,各个元素之间用“,”分开。例如:

```
int a[5] = {5,4,3,2,1};
```

即 $a[0]=5$ 、 $a[1]=4$ 、 $a[2]=3$ 、 $a[3]=2$ 、 $a[4]=1$ 。

初始化数组时也应注意以下几点。

(1)若{ }中初值的个数小于数组元素个数,即只有部分数组元素赋给了初值,后部分没有赋给初值的元素则置以相应类型的默认值(如 int 赋 0, char 型赋“\0”, float 型赋 0.000000 等)。例如:

```
int a[5] = {5,4,3};
```

则有 $a[0]=5$ 、 $a[1]=4$ 、 $a[2]=3$ 、 $a[3]=0$ 、 $a[4]=0$ 。

(2)若{ }中初值的个数=数组元素个数,则在数组定义时可省略元素个数,此时数组长度由{ }中的个数来确定。例如:

```
int a[] = {5,4,3,2,1};
```

相当于: $\text{int } a[5] = \{5,4,3,2,1\};$

(3)在{ }中定义的初值的个数不能大于数组元素的个数。例如:

```
int a[5] = {6,5,4,3,2,1};
```

是错误的,编译时不会通过。

6.1.2 一维数组元素的引用

1. 一维数组的引用

C 语言规定数组不能以整体形式参与各种运算,参与各种运算的只能是数组元素,即在程序中不能一次引用整个数组而只能逐个引用数组元素。一维数组的引用形式为:

数组名[下标]

其中,下标可以是整型常量、整型变量或整型表达式。例如:

```
int a[5] = {5,4,3,2,1};
```

则其元素的引用形式为: $a[0]=5$ 、 $a[1]=4$ 、 $a[2]=3$ 、 $a[3]=2$ 、 $a[4]=1$ 。

一个数组元素可以像一般变量那样参与赋值、算术运算、输入/输出等操作。所以,一维数组的元素可称为单下标变量。

2. 一维数组的输入/输出

可以通过 scanf() 和 printf() 语句为数组元素赋值和输出数组元素的值。

```
int a[50], i;
```

```
for(i=0; i<=49; i++)
```

```
scanf("%d", &a[i]); //从键盘为数组元素赋值
```

```
for(i=0; i<=49; i++)
```

```
printf("%d", a[i]); //输出数组元素的值
```

注意:不能用 scanf("%d", &a) 和 printf("%d", a) 对数组进行操作。

【例 6.1】 输入 10 个整数,求其中最大值和最小值。

【解题思路】

定义一个数组用来存储 10 个整数,变量 i 表示数组下标,定义两个变量 max 和 min 用来存储最大值和最小值,它们的初始值用数组元素 $a[0]$ 表示,采用 for 循环逐个比较数组元素

$a[i]$ 与 \max 、 \min , 把小的值放在 \min 中, 大的值放在 \max 中。

【参考代码】

```
#include <stdio.h>

void main()
{
    int i, max, min, a[10];
    printf("请输入 10 个数:");
    for(i=0; i<10; i++)
        scanf("%d", &a[i]);           //输入 10 个整数到数组 a 中
    max = min = a[0];                 //假定 max 和 min 都等于 a[0]
    for(i=0; i<10; i++)
    {
        if(a[i] > max) max = a[i];
        if(a[i] < min) min = a[i];
    }
    printf("最大值: %d, 最小值: %d\n", max, min);
}
```

程序运行结果如图 6.2 所示。

【例 6.2】 输入 10 个整数, 按由小到大排序, 要求使用选择排序法。

【解题思路】

选择排序过程: 外层循环用 i 变量控制 (i 从 0 到 $N-2$) 指向数组的一个元素, 内层循环用 j 控制, 用 \min 变量指向从 $i+1$ (从 $i+1$ 到 $N-1$) 位置开始的后边的最小的一个元素 (选择到的最小的元素), 使之与 i 位置的元素交换。

显然, 对于有 n 个元素的数组, 也需要 $n-1$ 趟排序。关键问题是如何使每趟排序能够选出一个最小的元素, 而不必交换多次。

【参考代码】

```
#include <stdio.h>

#define N 10

void main()
{
    int i, j, min, a[N], temp;
    printf("请输入 %d 个数:", N);           //提示输入行
    for(i=0; i<N; i++)
        scanf("%d", &a[i]);                 //输入 10 个整数到数组 a 中
    printf("排序前:");
    for(i=0; i<N; i++)
        printf("%d ", a[i]);               //输出排序前的 10 个数
    printf("\n");
    for(i=0; i<N-1; i++)                   //排序
```



图 6.2 例 6.1 运行结果

```

    {
        min = i;
        for(j = i + 1; j < N; j++)
            if(a[min] > a[j])
                min = j;
        { temp = a[i];
          a[i] = a[min];
          a[min] = temp;
        }
    }

    printf("排序后:");
    for(i = 0; i < N; i++)
        printf("%d ", a[i]);
    printf("\n");
}

```

//min 总是最小元素的下标
//把 min 位置元素交换到第 i 位置
//输出排序后的 10 个数

程序运行结果如图 6.3 所示。



图 6.3 例 6.2 运行结果

【注意点】

(1) 程序片断:

```
for(i = 0; i < N - 1; i++)
```

```

{
    min = i;
    for(j = i + 1; j < N; j++)
        if( a[min] > a[j])
            min = j;
    {
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
}

```

不能书写成如下片断:

```

for(i = 0; i < N - 1; i++)
{
    for(j = i + 1; j < N; j++)
        if(a[i] > a[j])

```



```
temp = a[i];
a[i] = a[j];
a[j] = temp;
```

```
}
}
```

后者不管 $a[i]$ 的大小如何,只要 $a[j]$ 的值比它小就要交换,这样频繁交换会影响程序运行的速度,这是不可取的;前者在找到一个最小的值之后才做交换操作,程序效率高。

(2) 数组的长度采用符号常量 `#define N 10`, 注意“`#define N 10`”不能写成“`#define N 10;`”,即尾部没有分号“`;`”。

【例 6.3】 在一个有序的数组中,插入一个数,使之插入后仍有序。

【解题思路】

首先在数组中找到合适的插入位置,然后将该位置后面的元素依次向后移动一个位置,这样就为这个要插入的数留出位置,最后将要插入的数保存到该位置。

【参考代码】

```
#include <stdio.h>
#define N 10
void main()
{
    int i, j;
    int a[N + 1] = {12, 23, 34, 45, 56, 67, 78, 89, 101, 110}, x;
    printf("插入前:");
    for(i = 0; i < N; i++) //输出插入前的 N+1 个数
        printf("%4d", a[i]);
    printf("\n");
    printf("请输入要插入的数:");
    scanf("%d", &x);
    for(i = 0; i < N; i++) //查出第一个大于要插入数的位置
    {
        if(a[i] > x)
            break;
    }
    for(j = N; j > i; j--) //为要插入的数留出位置
    {
        a[j] = a[j - 1];
    }
    a[i] = x; //将要插入的数保存到第 i 位置
    printf("插入后:");
    for(i = 0; i < N + 1; i++) //输出插入后的 N+1 个数
        printf("%4d", a[i]);
    printf("\n");
}
```

程序运行结果如图 6.4 所示。

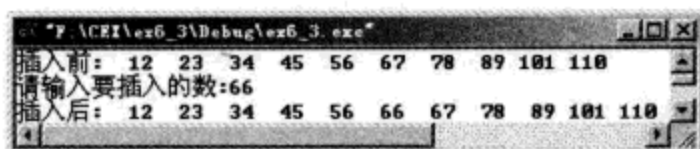


图 6.4 例 6.3 运行结果

【注意点】

- (1) 初始化数组时没有给出最后一个元素的值,但在插入前输出最后一个元素值是 0。
- (2) 删除数组中的一个元素与插入一个元素相似,请读者思考其方法。

【例 6.4】 采用二分查找法查找数组中是否存在一个数,并确定其位置。

【解题思路】

使用二分法查找数的前提是,数组中的数据必须是有序的(本例假定数组中数据是升序的)。顺序查找(从数组首个元素开始依次查找的方法)在数据量较小的情况下应用较普遍,当数据量很大时,用二分法可明显加快查找速度:若有 n 个数且 $2^m = n$,用二分法最多只需找 $m+1$ 次,如 $n=128(2^7)$,则最多找 8 次。

采用二分查找法算法步骤如下:

- (1) 首先确定一个查找范围: $low = 0; high = n - 1$;
- (2) 确定中间元素的位置: $mid = (low + high) / 2$;
- (3) 用中间位置上的元素 $a[mid]$ 与待查找的值 x 进行比较;
若 $a[mid] = x$,则找到,输出下标;
若 $a[mid] > x$,则说明 x 在查找区间的前半部,缩小查找范围,令 $high = mid - 1$;
若 $a[mid] < x$,则说明 x 在查找区间的后半部,缩小查找范围,令 $low = mid + 1$;
- (4) 重复(2)和(3),直到 $low > high$ 为止。

【参考代码】

```
#include <stdio.h>
void main()
{
    int a[] = {12,23,34,45,56,67,78,89,101,110};
    int i, low, high, mid, x;
    printf("初始化数组如下:\n");
    for(i=0; i<10; i++)
        printf("%5d", a[i]);
    printf("\n");
    printf("请输入查找的数据:");
    scanf("%d", &x);
    high = 9;
    low = 0;
    while(low <= high)
    {
        mid = (high + low) / 2;
        if (x == a[mid])
```

```

    printf("该数已找到,%d 是第%d 个数.\n",x,mid+1);
    break;
}
if (x > a[mid] )
    low = mid + 1;
else
    high = mid - 1;
if( low > high)
    printf("该数没有找到!\n");

```

程序运行结果如图 6.5 所示。

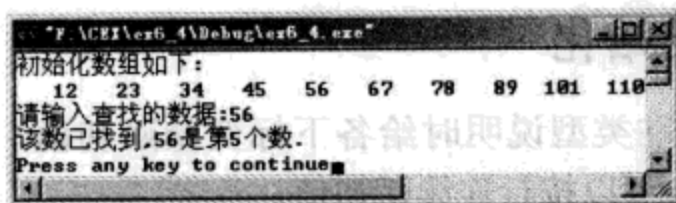


图 6.5 例 6.4 运行结果

6.2 二维数组

6.2.1 二维数组的定义

前面介绍的数组只有一个下标,称为一维数组。在实际应用中很多情况都需要二维或多维数组。即含有多个下标,以标识它在数组中的位置。含有两个下标的数组称为二维数组,其中的数据是按二维表的形式排列,即数据分为若干行和列。C 语言规定,二维数组的行下标和列下标也是从 0 开始计算的。

二维数组定义的一般形式为:

类型说明符 数组名[常量表达式 1][常量表达式 2];

其中常量表达式 1 表示行下标的长度,常量表达式 2 表示列下标的长度。例如:

```
int a[3][4];
```

定义了一个 3 行 4 列的数组,数组名为 a,数组元素的数据类型为 int,该数组的元素共有 $3 \times 4 = 12$ 个,即:

```

a[0][0]  a[0][1]  a[0][2]  a[0][3]
a[1][0]  a[1][1]  a[1][2]  a[1][3]
a[2][0]  a[2][1]  a[2][2]  a[2][3]

```

对此二维数组可以理解为有三个元素:a[0]、a[1]、a[2],这三个元素每个都是包含 4 个元素的数组,如图 6.6 所示。

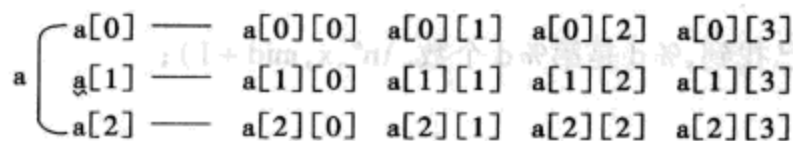


图 6.6 数组 a 的存储结构

数组 $a[3][4]$ 在内存的存放顺序如图 6.7 所示。从图中可看出先存放 $a[0]$ 中的四个元素 $a[0][0]$, $a[0][1]$, $a[0][2]$, $a[0][3]$; 后存放 $a[1]$ 中四个元素 $a[1][0]$, $a[1][1]$, $a[1][2]$, $a[1][3]$; 最后才存放 $a[2]$ 中的四个元素 $a[2][0]$, $a[2][1]$, $a[2][2]$, $a[2][3]$ 。

从图中还可以看出, $a[0]$ 、 $a[1]$ 、 $a[2]$ 代表的不是数组的具体元素, 不能当作带有下标的变量一样使用, 它们每个都相当于一个一维数组, 即一维数组名, 是代表数组中三行元素的首地址。

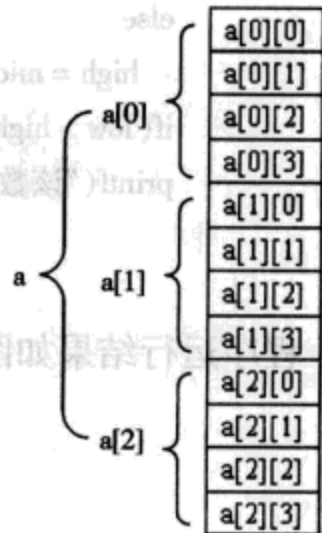


图 6.7 数组 a 在内存中的存放位置

6.2.2 二维数组的初始化

二维数组的初始化也是在类型说明时给各下标变量赋初值。二维数组可按行分段赋初值, 也可以按行连续赋初值。

例如对数组 $a[3][4]$ 的赋值有以下两种。

(1) 按行分段赋初值。

```
int a[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

(2) 按行连续赋初值。

```
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

这两种赋初值是完全一样的。

二维数组赋初值应注意如下几点。

(1) 可以只对部分元素赋初值, 未赋初值的元素自动取 0 (int 为 0, float 为 0.000000, char 为 '\0')。

例如 $\text{int } a[3][4] = \{\{1\}, \{2\}, \{3\}\};$ 是对每一行的第一列元素赋初值, 未赋初值的元素取值为 0。即相当于:

```
int a[3][4] = {{1,0,0,0}, {2,0,0,0}, {3,0,0,0}};
```

而

```
int a[3][4] = {{0,1}, {2}, {0,0,3}};
```

则相当于:

```
int a[3][4] = {{0,1,0,0}, {2,0,0,0}, {0,0,3,0}};
```

(2) 如果赋全部初值, 则行下标的长度可以不给出。例如:

```
int a[][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

```
int a[][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

6.2.3 二维数组元素的引用

二维数组的元素也称为双下标变量, 其一般表示形式为:

数组名[下标 1][下标 2]

其中, 下标 1 称为左下标或行下标, 下标 2 称为右下标或列下标。下标应为整型常量或整型表

达式。例如:`a[2][3]`。

但要注意的是,下标变量与数组定义中的数组的长度含义不同。数组定义中方括号中给出的是某维的长度,即可取下标的最大值;而数组元素中的下标是该元素在数组中的位置标识。前者是常量,后者可以是常量、变量和表达式(必须有具体的值)。

【例 6.5】 一个班级有一个小组,某学期某门课程考试分理论成绩和实验成绩,如表 6.2 所示。

表 6.2 一组学生成绩

姓名	理论成绩	实验成绩
张三	67	89
李四	78	90
王五	76	56

求该小组理论和实验的平均成绩及每名学生的总分。

【解题思路】

要想存放成绩信息必须定义一个二维数组,用来存放理论成绩、实验成绩及总分。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int i,j,a[3][3];
```

```
float average1,average2;
```

```
average1 = 0;average2 = 0;
```

```
printf("请输入 3 名同学的理论与实验成绩:\n");
```

```
for(i=0;i<3;i++)
```

```
for(j=0;j<2;j++)
```

```
scanf("%d",&a[i][j]);
```

```
printf("3 名同学的理论与实验成绩:\n");
```

```
for(i=0;i<3;i++)
```

```
{
```

```
for(j=0;j<2;j++)
```

```
printf("%d",a[i][j]);
```

```
printf("\n");
```

```
}
```

```
for(i=0;i<3;i++)
```

```
{
```

```
a[i][2] = a[i][0] + a[i][1];
```

```
average1 = average1 + a[i][0];
```

```
average2 = average2 + a[i][1];
```

```
}
```

```
printf("%8s %8s %8s\n","理论成绩","实验成绩","总分");
```



```
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
        printf("%8d",a[i][j]);
    printf("\n");
}
printf("理论平均分:%4.1f,实验平均分:%4.1f\n",average1/3,average2/3);
```

程序的运行结果如图 6.8 所示。

【注意点】

注意输出数据的对齐操作。

6.2.4 多维数组的定义

上面详细讲解了二维数组知识,多维数组就不再多叙述了,这里只简单介绍多维数组的概念。

二维数组实际上是一种最简单的多维数组。C 语言允许使用高于二维的多维数组。多维数组定义的一般形式为:

类型说明符 数组名[常量表达式 1][常量表达式 2]……[常量表达式 n];

例如:

```
float a[3][4][5];
```

以上定义了一个三维数组,可以理解为:三维数组 a 包含三个二维数组(a[0]、a[1]和 a[2]),每个二维数组又包含了 4 个一维数组(如 a[0][1]、a[1][1]),而每个一维数组包含 5 个 float 型的数组元素(如 a[0][0][1]和 a[0][1][1])。该数组可用表 6.3 表示。

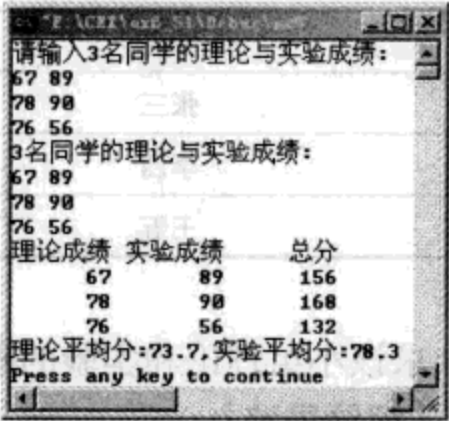


图 6.8 例 6.5 的运行结果

表 6.3 a[3][4][5]的组成

三维 数组名	二维 数组名	一维 数组名	数组元素
a	a[0]	a[0][0]	a[0][0][0],a[0][0][1],a[0][0][2],a[0][0][3],a[0][0][4]
		a[0][1]	a[0][1][0],a[0][1][1],a[0][1][2],a[0][1][3],a[0][1][4]
		a[0][2]	a[0][2][0],a[0][2][1],a[0][2][2],a[0][2][3],a[0][2][4]
		a[0][3]	a[0][3][0],a[0][3][1],a[0][3][2],a[0][3][3],a[0][3][4]
	a[1]	a[1][0]	a[1][0][0],a[1][0][1],a[1][0][2],a[1][0][3],a[1][0][4]
		a[1][1]	a[1][1][0],a[1][1][1],a[1][1][2],a[1][1][3],a[1][1][4]
		a[1][2]	a[1][2][0],a[1][2][1],a[1][2][2],a[1][2][3],a[1][2][4]
		a[1][3]	a[1][3][0],a[1][3][1],a[1][3][2],a[1][3][3],a[1][3][4]
	a[2]	a[2][0]	a[2][0][0],a[2][0][1],a[2][0][2],a[2][0][3],a[2][0][4]
		a[2][1]	a[2][1][0],a[2][1][1],a[2][1][2],a[2][1][3],a[2][1][4]
		a[2][2]	a[2][2][0],a[2][2][1],a[2][2][2],a[2][2][3],a[2][2][4]
		a[2][3]	a[2][3][0],a[2][3][1],a[2][3][2],a[2][3][3],a[2][3][4]

该数组在内存中的存储顺序是先行后列,与二维数组类似。

6.3 字符数组

用来存放字符的数组称为字符数组。字符数组用来保存字符序列或文本等。由于人们常用 C 语言编写处理字符序列或文本的程序,它为处理字符数组提供了支持。

6.3.1 字符数组的定义

字符数组的定义形式与之前介绍的定义数组相同。例如:

```
char str[7];           //定义 str 为字符数组名,7 为字符数组长度
```

```
str[0] = 's'; str[1] = 't'; str[2] = 'u'; str[3] = 'd'; str[4] = 'e'; str[5] = 'n'; str[6] = 't';
```

定义了一个字符数组,包含 7 个数组元素,在赋值后数组的存储情况如图 6.9 所示。

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]
s	t	u	d	e	n	t

图 6.9 字符数组的存储表示

注意:字符型与整型可以通用,但由于 int 在 Visual C++ 6.0 中占用 4 个字节,而字符仅占用 1 个字节,所以用 int str[7] 来定义一个字符数组虽然是合法的,但浪费存储空间,所以一般不提倡使用整型数组来定义字符数组。

6.3.2 字符数组的初始化

对字符数组初始化,也允许在类型说明时对其作初始化赋值。例如:

```
char str[7] = {'s', 't', 'u', 'd', 'e', 'n', 't'};
```

当对全体元素赋初值时也可以省略长度说明。例如:

```
char str[] = {'s', 't', 'u', 'd', 'e', 'n', 't'};
```

这里字符数组 str 的长度为 7。

字符数组初始化需注意以下几点。

(1) 逐个元素初始化。

```
char str[7] = {'s', 't', 'u', 'd', 'e', 'n', 't'};
```

(2) 初始化如果少于数组长度,多余的元素自动填充“空”('\0', ASCII 值为 0),如图 6.10 所示。

```
char str[10] = {'s', 't', 'u', 'd', 'e', 'n', 't'};
```

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]	str[9]
s	t	u	d	e	n	t	\0	\0	\0

图 6.10 一个字符数组的存储表示

(3) 指定初值时,若未指定数组长度,则长度等于初值个数。

```
char str[] = {'s', 't', 'u', 'd', 'e', 'n', 't'};
```

(4) 给定数组长度小于初始化字符个数,则编译时出错。

6.3.3 字符串与字符数组

字符串是一组字符,这些字符在内存中连续分布,在最后一个字节单元中用'\0'表示结束。这种结构与字符数组十分相似,实际上字符数组可以用来表示和存储字符串。例如:

```
char str[] = {'s', 't', 'u', 'd', 'e', 'n', 't', '\0'};
```

数组 str 中存储的就是字符串"student",在定义字符数组时,可以把字符串直接赋值给字符数组。例如:

```
char str[] = "student";
```

或

```
char str[] = {"student"};
```

这种定义方法与

```
char str[] = {'s', 't', 'u', 'd', 'e', 'n', 't', '\0'};
```

是完全等价的。

【例 6.6】字符串与字符数组操作。

【解题思路】

本题目关键是对字符串结束标志操作。

```
#include <stdio.h>
```

```
void main()
```

```
{
    char str1[] = "student";
```

```
    char str2[] = {'s', 't', 'u', 'd', 'e', 'n', 't', '\0'};
```

```
    char str3[] = {'s', 't', 'u', 'd', 'e', 'n', 't'};
```

```
    int i;
```

```
    printf("str1 长度 = %d", sizeof(str1));
```

```
    for(i = 0; i < sizeof(str1); i++)
```

```
        printf("%02x", str1[i]);
```

```
    printf("\n");
```

```
    printf("str2 长度 = %d", sizeof(str2));
```

```
    for(i = 0; i < sizeof(str2); i++)
```

```
        printf("%02x", str2[i]);
```

```
    printf("\n");
```

```
    printf("str3 长度 = %d", sizeof(str3));
```

```
    for(i = 0; i < sizeof(str3); i++)
```

```
        printf("%02x", str3[i]);
```

```
    printf("\n");
```

```
    printf("str3 长度 = %d", sizeof(str3));
```

```
    for(i = 0; i < sizeof(str3); i++)
```

```
        printf("%2c", str3[i]);
```

```
    printf("\n");
}
```

程序运行结果如图 6.11 所示。

6.3.4 字符串的输入/输出

前面已经讨论过字符串的输出在 `printf()` 函数中用“%s”来进行,字符串用字符数组来表示,因此 `printf()` 函数用“%s”来输出字符数组中所表示的字符串。

同样 `scanf()` 函数用“%s”来输入一个字符串,输入的字符串存储在指定的字符数组中。例如:

```
char s[10];
scanf("%s",s);
```

注意在 `scanf()` 函数中数组名称前面没有“&”符号, `scanf()` 在输入字符串时会到输入流中去读取。读取的规则如下。

(1) 跳过前面的空格,从一个非空格字符开始,一直取出连续的非空格字符,直至遇到空格或回车符为止,把这些字符读取到字符数组,并在最后加结束标志“\0”。

(2) 如果字符串的长度超过字符数组的容量,则 C 程序不会自动停止读取字符,程序设计人员应设法保证读取的字符数不要超过字符数组的容量,不然数组越界会带来意想不到的错误。

【例 6.7】 字符串的读取与输出。

【参考代码】

```
#include <stdio.h>
void main()
{
    char str[25];
    printf("请输入字符串:");
    scanf("%s",str);
    printf("%s\n",str);
}
```

程序运行结果为如图 6.12 所示。

【注意点】

在输入的字符串中读取“this”后遇到了空格,因此只读取了“this”。

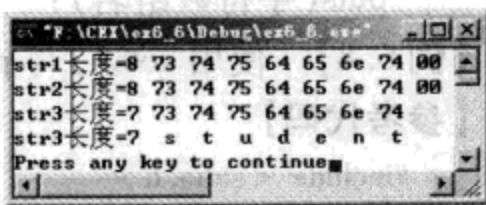


图 6.11 例 6.6 的运行结果

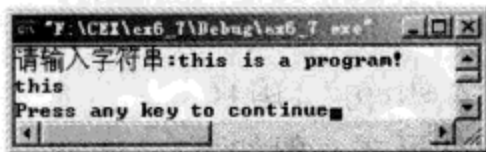


图 6.12 字符串读取与输出

6.3.5 字符串输入/输出函数

1. gets() 函数

`gets()` 是用来输入字符串的函数,其一般形式为:

```
gets(字符数组名);
```

`gets()` 从输入流中读取一行字符,一直读到“\n”,但不包括“\n”字符,最后在字符串末尾自动加上的结束标志“\0”。如果字符串的长度超过字符数组的容量,则 C 程序不会自动停止读取字符,会出现意想不到的错误,程序设计人员应该注意此点。

2. puts() 函数

`puts()` 是用来输出字符串的函数,其一般形式为:

puts(字符数组名);

【例 6.8】 使用 gets() 与 puts() 输入/输出一串字符。

【参考代码】

```
#include <stdio.h>

void main()
{
    char str[25];
    printf("请输入字符串:");
    gets(str);
    puts(str);
}
```

程序运行结果如图 6.13 所示。

【注意点】

“puts(str);”中没有“\n”符,但输出一串字符后,会自动换行。

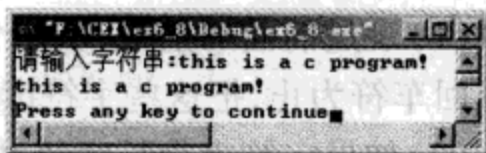


图 6.13 字符串读取与输出

6.4.6 常用字符串处理函数

字符串在程序中大量使用,关于字符串有一些常用的函数,这些函数在 string.h 头文件中说明,在使用时程序的开始部分要包含下列语句:

```
#include <string.h>
```

字符串处理函数大致可分为字符串的输入、输出、合并、修改、比较、复制等类型。

1. strlen() 函数

其形式一般如下:

```
strlen(字符数组);
```

strlen() 称为字符串长度函数,它用来计算字符串的实际长度,即从第一个字符开始一直读取到“\0”前的一个字符串的总字符数。例如:

```
char str[] = "student";
strlen(str);
```

2. strcpy() 函数

其形式一般如下:

```
strcpy(字符数组 1, 字符数组 2);
```

其功能是把字符数组 2 的字符串复制到字符数组 1 中去(即给一个字符数组赋值)。

例如:

```
char str2[] = "student", str1[10];
strcpy(str1, str2);
```

注意:

(1) “字符数组 1”必须是字符数组名称;

(2) 串结束标志也一同复制;

(3) “字符数组 2”可以是字符串常量,也可以是字符数组名称。

3. strcat() 函数

其形式一般如下:

strcat(字符数组1,字符数组2);

其功能是把字符数组2中的字符串连接到字符数组1中字符串后面,但字符数组1后的结束标志被字符数组2第一个字符覆盖了。例如:

```
char str2[] = "student", str1[20] = "I am a ";
```

```
strcat(str1, str2);
```

合并后的字符串 str1 如图 6.14 所示。

I		a	m		a		s	t	u	d	e	n	t	\0					
---	--	---	---	--	---	--	---	---	---	---	---	---	---	----	--	--	--	--	--

图 6.14 字符串连接

3. strcmp() 函数

其形式一般如下:

strcmp(字符数组1,字符数组2);

其功能是按 ASCII 顺序比较两个字符数组中的字符串,并由函数返回值返回比较结果。

(1) 字符串1 = 字符串2, 返回值为0;

(2) 字符串1 > 字符串2, 返回值 > 0;

(3) 字符串1 < 字符串2, 返回值 < 0;

【例 6.9】字符串处理函数的使用。

【参考代码】

```
#include <stdio.h>
```

```
#include <string.h>
```

```
void main()
```

```
{
```

```
char str1[25], str2[10];
```

```
int k;
```

```
printf("请输入第1字符串:");
```

```
gets(str1);
```

```
fflush(stdin);
```

```
printf("请输入第2字符串:");
```

```
gets(str2);
```

```
printf("输出的两个字符串为:\n");
```

```
puts(str1);
```

```
puts(str2);
```

```
printf("str1 长度 = %d\n", strlen(str1));
```

```
printf("str2 长度 = %d\n", strlen(str2));
```

```
k = strcmp(str1, str2);
```

```
printf("str1 与 str2 比较结果为:");
```

```
if(k > 0)
```

```
printf("str1 > str2\n");
```

```
if(k == 0)
```

```
printf("str1 = str2\n");
```

```
if(k < 0)
```

```
printf("str1 < str2\n");
```

```
printf("str2 连接到 str1 后 str1 为:%s\n",strcat(str1,str2));
printf("str2 复制到 str1 后 str1 为:%s\n",strcpy(str1,str2));
```

程序的运行结果如图 6.15 所示。

【注意点】

(1) 字符串 str1 中最后还有一个空格也算字符串中的字符。

(2) 字符串 str2 中最前还有一个空格也算字符串中的字符。

(3) fflush(stdin); 为清除键盘缓冲区。

(4) 输入字符串时不提倡使用 scanf() 函数, 因为 scanf() 遇到空格就会停止读取字符串了。

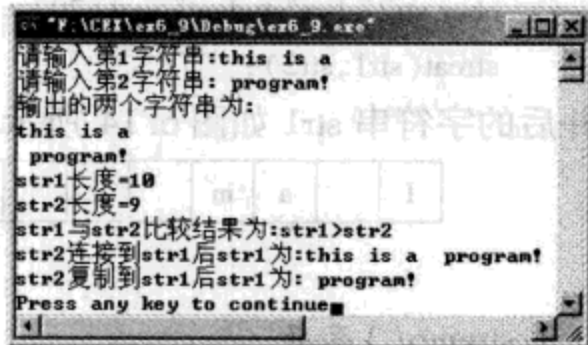


图 6.15 字符串处理函数

6.4 综合实例

【例 6.10】 输入一行字符, 统计其中有多少个单词, 单词之间用空格分开。

【解题思路】

定义字符数组 str 用来保存一行字符, count 用来统计单词个数, isword 用来判断前一个字符是否为非空字符。前面一个字符是否为空格可以从 isword 的值看出来, 若 isword = 0, 则表示前一个字符是空格; 如果 isword = 1, 意味着前一个字符为非空格。

单词的数目可以由空格出现的次数决定(连续若干个空格作为出现一次空格; 一行开头的空格不统计在内)。如果测出某一个字符非空格, 而它的前面的字符是空格, 则表示“新的单词开始了”, 这时单词数量才能加 1。isword 用来判断单词是否开始, 如果连续空格则 isword 仍为 0, 当遇到一个字符, 而前面是空格(isword = 0)时 count 才能加 1。如果当前字符非空格而其前面的字符也非空格, 则意味着仍然是原来的那个单词的继续, count 不应加 1。

【参考代码】

```
#include <stdio.h>

void main()
{
    char str[81];
    int i, count = 0, isword = 0;
    char ch;
    printf("请输入字符串 str:");
    gets(str);
    fflush(stdin);
    printf("输入的字符串 str 为:\n");
    puts(str);
    for(i = 0; (ch = str[i]) != '\0'; i++)
        if(ch == ' ')
            isword = 0;
```

```

else if(isword == 0)
{
    isword = 1;
    count ++;
}
printf("字符串 str 共有%d 个单词!\n",count);
}

```

程序运行结果如图 6.16 所示。

【注意点】

输入的字符串以回车结束,在把输入的字符串存入字符数组时,把回车符转化成“\0”(字符串结束标志)存储在字符数组中。

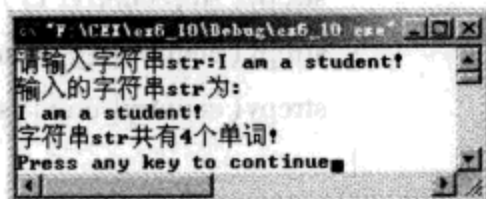


图 6.16 例 6.10 运行结果

【例 6.11】 输入 5 个国家的名称并按字母顺序排列输出。

【解题思路】

5 个国家名称应由一个二维字符数组来处理。然而 C 语言规定,可以把一个二维数组当成多个一维数组。因此,本题又可以按 5 个一维数组处理,每一个一维数组就是一个国家名称字符串。用字符串比较函数比较各个一维数组的大小,并排序,输出结果。

本例的字符串比较采用选择排序法(前面已经介绍过),定义变量 min,其值表示最小的那个字符串的位置,把 min 指向的最小字符串与第 i 个位置的字符串比较,如果 min 位置的字符串大,则交换两个字符串,交换的方法采用字符串复制函数 strcpy()。

【参考代码】

```

#include <stdio.h>
void main()
{
    char str[30],country[5][30];
    int i,j,min;
    for(i=0;i<5;i++)
    {
        printf("请输入第%d 国家名称:",i);
        gets(country[i]);
        fflush(stdin);
    }
    printf("\n 输入的国家名称为:\n");
    for(i=0;i<5;i++)
    {
        puts(country[i]);
    }
    for(i=0;i<4;i++)
    {
        min = i;

```

```

strcpy(str, country[i]);
for(j = i + 1; j < 5; j++)
    if(strcmp(country[j], str) < 0)
    {
        min = j;
        strcpy(str, country[j]);
    }
strcpy(str, country[i]);
strcpy(country[i], country[min]);
strcpy(country[min], str);
printf("排序后的国家名称:\n");
for(i = 0; i < 5; i++)
    puts(country[i]);
}

```

程序运行结果如图 6.17 所示。

6.5 案例应用举例

【例 6.12】 利用循环和数组知识编写“图书管理系统”，实现“图书增加”子模块程序。

【参考代码】

```

#include <stdio.h>
#include <string.h>
#define N 2
void main()
{
    int booknum[N];
    char bookname[N][20];
    char bookauthor[N][20];
    char press[N][50];
    float price[N];
    int count[N];
    int i;
    for(i = 0; i < N; i++)
    {
        printf("请输入第%d本书的信息:\n", i + 1);
        printf("书号:");
        scanf("%d", &booknum[i]);
        fflush(stdin);
        printf("书名:");
        gets(bookname[i]);
    }
}

```



图 6.17 例 6.11 运行结果

```

fflush(stdin);
printf("作者:");
gets(bookauthor[i]);
fflush(stdin);
printf("出版社:");
gets(press[i]);
fflush(stdin);
printf("书价:");
scanf("%f",&price[i]);
fflush(stdin);
printf("剩余本数:");
scanf("%d",&count[i]);
fflush(stdin);
printf("输出录入的图书信息:\n");
printf("书号    书名    作者    出版社    书价    剩余本数\n");
for(i=0;i<N;i++)
{
    printf("%4d",booknum[i]);
    printf("%9s",bookname[i]);
    printf("%10s",bookauthor[i]);
    printf("%11s",press[i]);
    printf("%8.1f",price[i]);
    printf("%10d\n",count[i]);
}

```

程序运行结果如图 6.18 所示。

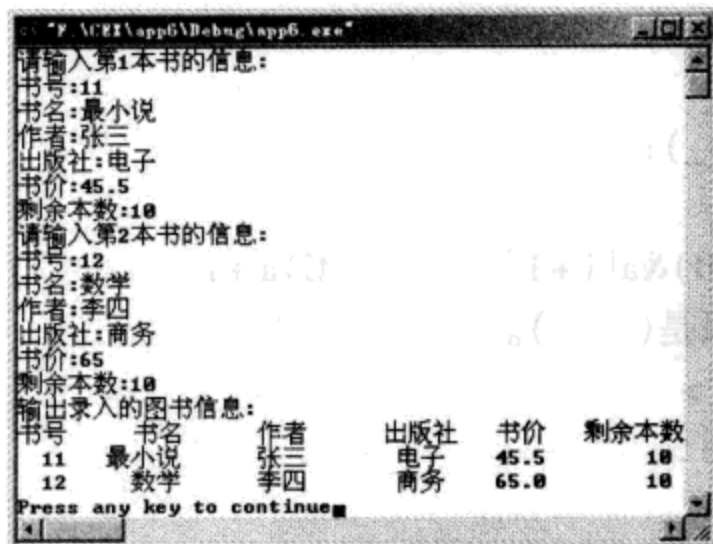


图 6.18 例 6.12 运行结果

本章小结

(1) 数组是程序设计中的重要内容,在实际应用中非常广泛。数组常常和循环一起使用,一维数组通常用单循环来完成,二维数组通常用双重循环来实现。

(2) 本章主要介绍一维数组、二维数组和字符数组的概念,以及它们的引用方式,要学会在实际中灵活运用。

习题六

一、选择题

1. 在 C 语言中,引用数组元素时,数组下标的类型允许是()。

A) 整型常量

B) 整型表达式

C) 整型常量或整型常量表达式

D) 任意类型的表达式

2. 以下程序段执行后,输出结果是()。

```
int k;
```

```
char s[] = {"ab\n\\012\\\\"};
```

//注意:"ab\n\\012\\\\"不能写成"ab\n\\012\\",否则编译错误。

```
k = strlen(s);
```

```
printf("%d\n",k);
```

A) 1

B) 8

C) 9

D) 10

3. 以下程序段数组所有元素都需要输入数据,应在下画线填入的是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int a[10], i = 0;
```

```
while(i < 10)
```

```
scanf("%d", _____);
```

```
}
```

A) a + (i++)

B) &a[i+1]

C) a + i

D) &a[++i]

4. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
#include <string.h>
```

```
void main()
```

```
{
```

```
int n[2] = {0}, i, j, k = 2;
```

```
for(i = 0; i < k; i++)
```

```
for(j = 0; j < k; j++)
```

```
n[j] = n[i] + 1;
```

```
printf("%d\n", n[k]);
}
```

A) 不确定的值

B) 3

C) 2

D) 1

5. 以下程序的输出结果是()。

```
#include <stdio.h>
void main()
{
    int a[] = {2, 3, 4, 5, 6, 7, 8, 9};
    int i, r = 1;
    for(i = 0; i <= 3; i++)
        r = r * a[i];
    printf("%d\n", r);
}
```

A) 720

B) 120

C) 24

D) 6

6. 当运行以下程序时输入三行, 每行都是从第一列开始, <CR> 代表 Enter 键;

a <CR>

b <CR>

cdef <CR>

则程序的输出结果是()。

#include <stdio.h>

#define N 6

void main()

```
{
    char c[N];
    int i = 0;
    for(i = 0; i < N; i++)
        c[i] = getchar();
    for(i = 0; i < N; i++)
        putchar(c[i]);
}
```

A) abcdef

B) a

C) a

D) a

b

b

b

c

cd

cdef

d

e

f

7. 当运行以下程序时输入三行, 每行都是从第一列开始, <CR> 代表 Enter 键;

a <CR>

b <CR>

```
cdef < CR >
```

则程序的输出结果是()。

```
#include <stdio.h>
#define N 6
void main()
{
    char c[N];
    int i=0;
    gets(c);
    printf("输出结果:\n");
    puts(c);
}
```

A) abcdef

B) a

C) a

D) a

b

b

c

cd

d

e

f

8. 当运行以下程序时输入三行,每行都是从第一列开始,<CR>代表 Enter 键;

```
a b cdef < CR >
```

则程序的输出结果是()。

```
#include <stdio.h>
void main()
{
    char c[6];
    int i=0;
    printf("请输入字符串 c:");
    scanf("%s",c);
    printf("输出结果:\n");
    printf("%s\n",c);
}
```

A) abcdef

B) a

C) a

D) a

b

b

c

cd

d

e

f

9. 当运行以下程序时输入三行,每行都是从第一列开始,<CR>代表 Enter 键;

```
a b cdef < CR >
```

则程序的输出结果是()。

```
#include <stdio.h>
#define N 6
void main()
{
    char c[N];
    int i=0;
    gets(c);
    printf("输出结果:\n");
    puts(c);
}
```

A) abcdef

B) a

C) a

b

c

d

e

f

D) a

b

c

d

10. 以下程序的输出结果是()。

```
#include <stdio.h>
void main()
{
    int i,x[3][3] = {1,2,3,4,5,6,7,8,9};
    for(i=0;i<3;i++)
        printf("%d",x[i][2-i]);
    printf("\n");
}
```

A) 1,5,9,

B) 1,4,7,

C) 3,5,7,

D) 3,6,9,

11. 以下能正确进行字符串定义、赋初值的语句组是()。

A) char s[5] = {'a','e','i','o','u'};

B) char *s; s="good!";

C) char s[5] = "good!";

D) char s[5]; s="good";

12. 有以下程序:

```
#include <stdio.h>
#include <string.h>
void main()
{
    char a[] = {'a','b','c','d','e','f','g','h','\0'};
    int i,j;
```

```

i = sizeof(a);
j = strlen(a);
printf("%d,%d\n", i, j);
}

```

程序运行后的输出结果是()。

A) 9, 9

B) 8, 9

C) 1, 8

D) 9, 8

13. 以下能正确定义一维数组的选项是()。

A) int a[5] = {0, 1, 2, 3, 4, 5};

B) char a[] = {'0', '1', '2', '3', '4', '5', '\0'};

C) char a = {'A', 'B', 'C'};

D) int a[5] = "0123";

14. 下面程序的运行结果是()。

```

#include <stdio.h>
void main()
{
    char a[] = "morning", t;
    int i, j = 0;
    for(i = 1; i < 7; i++)
        if(a[j] < a[i])
            j = i;
    t = a[j];
    a[j] = a[7];
    a[7] = a[j];
    puts(a);
}

```

A) mrgnir

B) mo

C) moring

D) morning

15. 现有如下程序:

```

#include <stdio.h>
void main()
{
    int k[30] = {12, 324, 45, 6, 768, 98, 21, 34, 453, 456};
    int count = 0, i = 0;
    while(k[i])
    {
        if(k[i] % 2 == 0 || k[i] % 5 == 0)
            count++;
        i++;
    }
    printf("%d,%d\n", count, i);
}

```


则程序的输出结果为()。

A) 7, 8

B) 8, 8

C) 7, 10

D) 8, 10

16. 当运行以下程序时,从键盘输入;AhaMA[空格]Aha <CR>,则下面程序的运行结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    char s[80], c = 'a';
```

```
    int i = 0;
```

```
    scanf("%s", s);
```

```
    while(s[i] != '\0')
```

```
    {
```

```
        if(s[i] == c)
```

```
            s[i] = s[i] - 32;
```

```
        else if(s[i] == c - 32)
```

```
            s[i] = s[i] + 32;
```

```
        i++;
```

```
    }
```

```
    puts(s);
```

```
}
```

A) ahAMa

B) AbAMa

C) AhAMa[空格]ahA

D) ahAMa[空格]ahA

17. 下面程序输出的结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int i;
```

```
    int a[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
    for(i = 0; i < 3; i++)
```

```
        printf("%d ", a[2 - i][i]);
```

```
        printf("\n");
```

```
}
```

A) 1 5 9

B) 7 5 3

C) 3 5 7

D) 5 9 1

18. 阅读下面程序,程序段的功能是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int c[] = {23, 1, 56, 234, 7, 0, 34}, i, j, t;
```

```

for(i = 1; i < 7; i++)
{
    t = c[i];
    j = i - 1;
    while(j >= 0 && t > c[j])
    {
        c[j + 1] = c[j];
        j--;
    }
    c[j + 1] = t;
}
for(i = 0; i < 7; i++)
    printf("%d ", c[i]);
putchar('\n');
}

```

A) 对数组元素的升序排列

B) 对数组元素的降序排列

C) 对数组元素的倒序排列

D) 对数组元素的随机排列

19. 以下不能对二维数组 a 进行正确初始化的语句是()。

A) int a[2][3] = {0};

B) int a[][3] = {{1, 2}, {0}};

C) int a[2][3] = {{1, 2}, {3, 4}, {5, 6}}; D) int a[][3] = {1, 2, 3, 4, 5, 6};

20. 以下程序输出的结果是()。

```

#include <stdio.h>
#include <string.h>
void main()
{
    char w[ ][10] = {"ABCD", "EFGH", "IJKL", "MNOP"}, k;
    for(k = 1; k < 3; k++)
        printf("%s\n", &w[k][k]);
}

```

A) ABCD

B) ABC

C) EFG

D) FGH

FGH

EFG

JK

KL

KL

IJ

0

M

21. 阅读下列程序, 程序的运行结果为()。

```

#include <stdio.h>
#include <string.h>
void main()
{

```

```

    char a[30] = "nice to meet you!";

```

```
strcpy(a + strlen(a)/2, "you");
printf("%s\n", a);
```

A) nice to meet you you

B) nice to

C) meet you you

D) nice to you

22. 有以下程序:

```
#include <stdio.h>
void main()
{
    int num[4][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}}, i, j;
    for(i=0; i<4; i++)
    {
        for(j=____; j<4; j++)
            printf("%4d", num[i][j]);
        printf("\n");
    }
}
```

若要按以下形式输出数组右上半三角

```
1  2  3  4
6  7  8
11 12
16
```

则在程序下画线处应填入的是 ()。

A) i-1

B) i

C) i+1

D) 4-i

二、填空题

1. 若有以下定义:

double w[10];

则数组元素下标的上限是_____, 下限是_____。

2. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main()
{
    int arr[10], i, k=0;
    for(i=0; i<10; i++)
        arr[i] = i;
    for(i=0; i<4; i++)
        k += arr[i] + i;
    printf("%d\n", k);
}
```

3. 若输入 3 个整数 3、2、1, 则以下程序的输出结果是 5 4 3。

```
#include <stdio.h>

void main()
{
    int i, n, a[10] = {0}; int t;
    scanf("%d%d%d", &n, &a[0], &a[1]);
    for(i = 1; i < n; i++)
    {
        t = a[i--];
        t += 3 * a[i];
        i++;
        if(t >= 10)
        {
            a[i++] = t/10;
            a[i] = t%10;
        }
        else
            a[i] = t;
    }
    for(i = 0; i <= n; i++)
        printf("%d", a[i]);
    printf("\n");
}
```

4. 以下程序的输出结果是 1 1 1。

```
#include <stdio.h>

void main()
{
    int i, j, row, col, m;
    int arr[3][3] = {{100, 200, 300}, {28, 72, -30}, {-850, 2, 6}};
    m = arr[0][0];
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
            if(arr[i][j] < m)
            {
                m = arr[i][j];
                row = i;
                col = j;
            }
    printf("%d, %d, %d\n", m, row, col);
}
```

5. 有下列程序,功能是把输入的十进制长整型数以十六进制的形式输出,完成程序。

```
#include <stdio.h>

void main()
{
    char b[17] = {"0123456789ABCDEF"};
    int c[5], d, i = 0, base = 16;
    long n;
    scanf("%ld", &n);
    do
    {
        c[i] = n % base;
        i++;
        n = _____;
    }
    while(n != 0);
    for( --i; i >= 0; --i)
    {
        d = _____;
        printf("%c", b[d]);
    }
    printf("\n");
}
```

6. 下面程序的功能是将一个字符串 str 的内容颠倒过来,请填空。

```
#include <stdio.h>
#include <string.h>

void main()
{
    int i, j, _____;
    char str[] = {"1234567"};
    for(i = 0, j = strlen(str) _____; i < j; i++, j--)
    {
        k = str[i];
        str[i] = str[j];
        str[j] = k;
    }
    printf("%s\n", str);
}
```

7. 下面程序的功能是从键盘输入一行字符,统计有多少个单词,单词间用空格分隔。补充所缺语句。


```
#include <stdio.h>
void main()
{
    char s[80], c1, c2 = ' ';
    int i = 0, num = 0;
    printf("请输入一串字符:\n");
    gets(s);
    while (s[i] != '\0')
    {
        c1 = s[i];
        if (i == 0) c2 = ' ';
        else c2 = s[i - 1];
        if (c1 != ' ' && c2 == ' ') num++;
        i++;
    }
    printf("There are %d words!\n", num);
}
```

8. 以下程序的输出结果是_____。

```
#include <stdio.h>
void main()
{
    int a[3][3] = {{1,2,9},{3,4,8},{5,6,7}}, i, s = 0;
    for(i = 0; i < 3; i++)
        s += a[i][i] + a[i][3 - i - 1];
    printf("%d\n", s);
}
```

9. 以下程序运行后的输出结果是_____。

```
#include <string.h>
#include <stdio.h>
void main()
{
    char ch[] = "abc", x[3][4]; int i;
    for(i = 0; i < 3; i++)
        strcpy(x[i], ch);
    for(i = 0; i < 3; i++)
        printf("%s", &x[i][i]);
    printf("\n");
}
```

三、编程题

1. 编写程序求任意方阵每行、每列、两对角线元素之和。
2. 请编程,判断一字符串是否是回文。若是回文,函数返回值为1,否则返回值为0。(回文是顺读和倒读都一样的字符串。)
3. 编写程序打印出以下形式的九九乘法表。

* * 一个乘法表 * *

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(1)	1	2	3	4	5	6	7	8	9
(2)	2	4	6	8	10	12	14	16	18
(3)	3	6	9	12	15	18	21	24	27
(4)	4	8	12	16	20	24	28	32	36
(5)	5	10	15	20	25	30	35	40	45
(6)	6	12	18	24	30	36	42	48	54
(7)	7	14	21	28	35	42	49	56	63
(8)	8	16	24	32	40	48	56	64	72
(9)	9	18	27	36	45	54	63	72	81

4. 调用随机函数,为一个 5×4 的矩阵置 100 以内的整数,输出该矩阵,求出每行元素之和,并把和值最大的那一行与第一行上的元素对调。若已定义 x 为 `int` 类型,调用随机函数步骤如下:

```
#include <stdlib.h>
```

```
.....
```

```
x = rand() % 100 /* 产生 0 到 100 的随机数 */
```

5. 有一个二维数组 `a[3][4]`,找出其中最大和最小的元素,并指出它们所在的行号和列号。要求输出结果格式如图 6.19 所示。



图 6.19 习题图

6. 打印出杨辉三角形。杨辉三角形中每一个元素是它肩上两个元素之和,要求打印如图 6.20 所示的形状。

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

```

图 6.20 习题图

第7章 函数

第1章已经介绍过,C语言源程序是由函数组成的。在前面各章介绍的C语言源程序中,都只有一个主函数 `main()`,但实际上程序往往由多个函数组成。

函数是C语言源程序的基本模块,通过对函数模块的调用实现特定的功能。C语言中的函数相当于其他高级语言的子程序。它不仅提供了极为丰富的库函数,而且还允许用户建立自己定义的函数。

在C语言中,所有的函数定义,包括主函数 `main()` 在内,都是平行的。也就是说,在一个函数的函数体内,不能再定义另一个函数,即不能嵌套定义。但是函数之间允许相互调用,也允许嵌套调用,习惯上把调用者称为主调函数。函数自己调用自己的情况,称为递归调用。

通过本章的学习,读者应该掌握以下内容:

- (1) 函数的概念;
- (2) 函数的参数及返回值;
- (3) 局部变量、全局变量和静态变量的概念及使用;
- (4) 函数的调用。

7.1 函数定义

在定义函数之前,先看一个例子。

【例7.1】求 $1+2+\cdots+n$ 的和,要求 n 通过键盘输入。

【参考代码】

方法一:

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int i,n;
```

```
    long sum=0;
```

```
    printf("请输入n的值:");
```

```
    scanf("%d",&n);
```

```
    for(i=1;i<=n;i++)
```

```
        sum=sum+i;
```

```
    printf("1+2+...+%d=%ld\n",n,sum);
```

```
}
```

方法二:

```
#include <stdio.h>
```

```
long sum(int n)
```

```
{
```

```

int s=0,i;
for(i=1;i<=n;i++)
    s=s+i;
return s;
}

void main()
{
    int n;
    printf("请输入 n 的值:");
    scanf("%d",&n);
    printf("1+2+...+%d=%ld\n",n,sum(n));
}

```

【程序说明】

方法一和方法二都实现了求 1 到 n 的和。方法一只使用了一个主函数 main(), 输入、计算和输出都在这个函数中完成。方法二用了一个主函数和一个子函数 sum(), 主函数的功能实现了输入和输出, 子函数的功能是计算 $1+2+\cdots+n$ 之和。主函数中仅用输出语句的输出项调用了 sum() 函数, 并把 n 的值传递给 sum()。sum() 在实现求 $1+2+\cdots+n$ 之和之后, 用了“return s;”把求得和 s 值返回给主函数的调用点 sum(n) 输出, 在 sum() 函数体中的 return 语句是把 s 的值作为函数的值返回给主调函数。有返回值的函数中至少应有一个 return 语句。

从本例可以看出, 如果主函数要求的功能非常复杂, 而且仅在主函数中实现其功能, 则查看、分析、调试程序都不清晰, 程序设计者很难理清自己设计的程序, 也很难进行团队合作。

函数一般由函数头和函数体两部分组成。函数头包括函数的类型、函数名、参数名及参数数据类型说明等。

函数的定义一般形式:

类型说明符 函数名(数据类型 形式参数 1, 数据类型 形式参数 2, ……)

函数体;

其中类型说明符和函数名称为函数头。类型说明符指明了本函数的类型。函数的类型实际上是函数返回值的类型。该类型说明符与前面介绍的各种说明符相同。函数名是由用户定义的标识符, 函数名后有一对圆括号, 即使其中无参数, 括号也不可少。

{ } 中的内容称为函数体。在函数体的声明部分, 是对函数体内部所用到的变量的类型说明。

在很多情况下都要求函数有返回值, 如果函数没有返回值, 则函数类型说明符写为 void。

在 C 程序中, 一个函数的定义可以放在任意位置, 既可放在主函数 main() 之前, 也可放在 main() 之后。

形式参数简称形参。形参是在函数定义时定义的, 主调函数中的参数是实际参数, 简称实参。例 7.1 中的 sum(n) 中的 n 是实参, sum(int n) 中的 n 就是形参。

【例 7.2】输入 3 个整数, 求其最大值。

【解题思路】

设 3 个整数为 a、b、c,要找出其中最大值,只要先找到 a 与 b 的最大值,设为 d,再把这个最大值 d 与 c 比较,找到 d 与 c 的最大值,设为 e,即可找到 a、b、c 中的最大值。在这个问题中两次用了在两个数中找最大值的方法,可以设计一个函数 max,它的定义为:

```
int max(int x,int y)
```

向 max() 提供两个参数 x 与 y,max 就可以返回 x 与 y 的最大值。max() 可以设计如下:

```
int max(int x,int y)
```

```
{  
    return x > y? x:y;  
}
```

其中 max() 是函数名,x 与 y 称为函数的形式参数,max() 前面的 int 表示函数 max() 的返回值类型,“return x > y? x:y;”是返回语句,它返回 x 与 y 的最大值。

【参考代码】

```
#include <stdio.h>
```

```
int max(int x,int y)
```

```
{  
    return x > y? x:y;  
}
```

```
void main()
```

```
{  
    int a,b,c,d,e;
```

```
    printf("请输入 a,b,c 的值:");
```

```
    scanf("%d%d%d",&a,&b,&c);
```

```
    d = max(a,b);
```

```
    e = max(d,c);
```

```
    printf("a,b,c 的最大值 = %d\n",e);  
}
```

程序运行结果如图 7.1 所示。

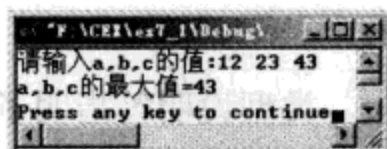


图 7.1 例 7.2 运行结果

7.2 函数的参数和函数的值

7.2.1 形式参数和实际参数

前面已经介绍过,函数的参数分为形参和实参两种。在本小节中,进一步介绍形参、实参的特点和两者的关系。形参出现在函数定义中,在整个函数体内都可以使用,离开该函数则不能使用。实参出现在主调函数中,进入被调函数后,实参变量也不能使用。形参和实参的功能

是作数据传送。发生函数调用时,主调函数把实参的值传送给被调函数的形参,从而实现主调函数向被调函数的数据传送。

函数的形参和实参具有以下特点。

(1) 形参是函数内部变量,只有在被调用时才分配内存单元,在调用结束后,即刻释放所分配的内存单元。因此,形参只有在函数内部有效。函数调用结束,返回主调函数后则不能再使用该形参变量。

(2) 实参可以是常量、变量、表达式、函数等,无论实参是何种类型的量,在进行函数调用时,它们都必须具有确定的值,以便把这些值传送给形参。因此应预先用赋值、输入等办法使实参获得确定值。

(3) 实参和形参在数量上、类型上、顺序上应严格一致,否则会发生“类型不匹配”的错误。

(4) 函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参,而不能把形参的值反向传送给实参。因此在函数调用过程中,形参的值发生改变,而实参中的值不会变化。

(5) 函数可以没有参数,但此时圆括号也不能少。

以例 7.1 为例。程序中定义了一个函数 `sum()`,该函数的功能是求 $\sum n$ 的值。在主函数中输入 `n` 值,并作为实参,在调用时传送给 `sum()` 函数的形参 `n` (注意,例子中的形参变量和实参变量的标识符都为 `n`,但这是两个不同的量,各自的作用域不同)。在主函数中用 `printf()` 语句输出一次 `n` 值,这个 `n` 值是实参 `n` 的值。从运行情况看,输入 `n` 值为 100。即实参 `n` 的值为 100。把此值传给函数 `sum()` 时,形参 `n` 的初值也为 100,在执行函数过程中,形参 `n` 的值变为 5050。返回主函数之后,输出实参 `n` 的值仍为 100。可见实参的值不随形参的变化而变化。

以例 7.2 为例。在例子中两次调用子函数 `max(int x, int y)`。

```
d = max(a, b);
```

```
e = max(d, c);
```

在第一次调用 `max()` 时,把变量 `a` 传递给 `x`,变量 `b` 传递给 `y`,其中 `x` 和 `y` 称为形参,`a` 和 `b` 称为实参。形参实际上就是函数内部的变量,`x` 和 `y` 就是 `max()` 中的内部形参变量,它们在内存中有自己的存储空间。当调用 `max()` 函数时,实参把它们传递给形参,或者说形参复制了实参的值,如图 7.2 所示。第二次调用的情况也相似。

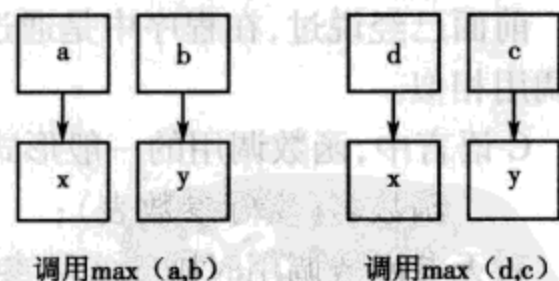


图 7.2 `max()` 的参数传递

7.2.2 函数的返回值

函数的值是指函数被调用之后,执行函数体中的程序段所取得的并返回给主调函数的值。如调用正弦函数取得正弦值,调用例 7.2 的 `max()` 函数取得最大值等。

对函数的值(或称函数返回值)有以下一些说明。

(1) 函数的值只能通过 `return` 语句返回主调函数。

`return` 语句的一般形式为:

```
return 表达式;
```

或者为:

return (表达式);

该语句的功能是计算表达式的值,并返回给主调函数。在函数中允许有多个 return 语句,但每次调用只能有一个 return 语句被执行,因此,只能返回一个函数值。

(2) 函数值的类型和函数定义中函数的类型应保持一致。如果两者不一致,则以函数类型为准,自动进行类型转换。

(3) 如函数值为整型,在函数定义时可以省去类型说明。

(4) 不返回函数值的函数,可以明确定义为“空类型”,类型说明符为“void”。如例 7.1 中函数 sum() 可以不向主函数返回函数值,因此可定义为:

```
void sum(int n)
```

```
{
    int i;
    long s = 0;
    for(i = 1; i <= n; i++)
        s = s + i;
    printf("1 + 2 + ... + %d = %ld\n", n, s);
}
```

一旦函数被定义为空类型后,就不能在主调函数中使用被调函数的函数值了。例如,在定义 sum() 为空类型后,在主函数中写下述语句:

```
printf("1 + 2 + ... + %d = %ld\n", n, s);
```

就是错误的。

为了使程序有良好的可读性并减少出错,凡不要求返回值的函数都应定义为空类型。

7.2.3 函数的调用

1. 函数调用的一般形式

前面已经说过,在程序中是通过对函数的调用来执行函数体的,其过程与其他语言的子程序调用相似。

C 语言中,函数调用的一般形式为:

函数名(实际参数表);

对无参函数调用时则无实际参数表。实际参数表中的参数可以是常数,变量或其他构造类型数据及表达式。各实参之间用逗号分隔。

2. 函数调用的方式

在 C 语言中,可以用以下几种方式调用函数。

(1) 函数表达式。函数作为表达式中的一项出现在表达式中,以函数返回值参与表达式的运算。这种方式要求函数是有返回值的。例如: $d = \max(x, y)$ 是一个赋值表达式,把函数 max() 的返回值赋予变量 d。

(2) 函数语句。函数调用的一般形式加上分号即构成函数语句。例如:

```
printf("%d", a);
scanf("%d", &b);
```

便是以函数语句的方式调用函数。

(3) 函数实参。函数作为另一个函数调用的实际参数出现。这种情况是把该函数的返回

值作为实参进行传送,因此要求该函数必须是有返回值的。例如:

```
printf("%d",max(x,y));
```

即是把 max() 调用的返回值又作为 printf 函数的实参来使用的。

7.2.4 函数原型

函数被定义之后,在调用之前一般需对该函数进行声明,以便向编译器指出该函数要使用什么样的格式或语法,也称为函数原型说明。函数原型说明位于程序开始处,位于头文件声明之后。

如果子函数在主函数 main() 之前定义,则不需要进行原型说明;如果它的主函数 main() 之后,则需要在主函数 main() 之前进行函数原型说明。

函数的原型说明在形式上与函数头类似,最后加上一个分号。原型说明中参数表里的参数名可以不写(只写参数类型)。即使在这里写参数名,所用名字也不必与函数定义用的名字一致。原型说明里的参数名可以起提示作用,也提倡给出有意义的名字,这将有利于函数的正确使用。

例 7.2 中的 max() 子函数在主函数 main() 之前,不需要进行原型说明,如果它的主函数 main() 之后,则要进行原型说明。

max() 的原型说明如下:

```
int max(int x,int y);
```

或

```
int max(int,int y);
```

7.2.5 函数的调用实例

【例 7.3】编写一个无参函数,实现如图 7.3 所示的图形。

```

      *
     ***
    *****
   *********
  ***********
 *****
  *****
   *****
    *****
     *****
      *****

```

图 7.3 例 7.3 效果图

【解题思路】

(1) 设置一个变量 rowcharcount 用来保存每行输出的字符数,后一行比前一行的字符数多 2, 所以用 rowcharcount += 2 实现每行字符数的变化;

(2) 在每行输出“*”之前,需要输出多个空格,用变量 j 表示输出的空格数和“*”字符数,j=6 表示输出空格的起始位置,每行输出的最多字符数为 rowcharcount。

【参考代码】

```

#include <stdio.h>

void display();

void main()

```

```
//display() 函数原型说明
```

```

printf("用\" * \"显示三角形:\n");
printf("-----\n");
display();
printf("\n");

void display() //定义函数 display()
{
    int i, j, rowcharcount = 0;
    for(i = 0; i <= 5; i++, rowcharcount += 2) //控制" * "字符数
    {
        for(j = 6; j > i; j--)
        {
            printf(" ");
        }
        for(j = 0; j <= rowcharcount; j++)
        {
            printf(" * ");
        }
        printf("\n");
    }
}

```

程序运行结果如图 7.4 所示。

【例 7.4】 根据用户的选择求不同形状的面积。

【解题思路】

在屏幕上给出提示菜单, 根据提示要求输入一个整数, 选择形状, 根据所选的形状, 给出相应的参数提示, 最后求出所选形状的面积。

【参考代码】

```

#include <stdio.h>
void AreaOfRectangular();
void AreaOfTriangle();
void AreaOfCircle();
void main()
{
    int choice;
    do
    {
        printf("面积函数:\n");
        printf("0、退出\n 1、长方形\n 2、三角形\n 3、圆形\n");
        printf("请选择功能:");
        scanf("%d", &choice);
    } while(choice < 0 || choice > 3);
}

```

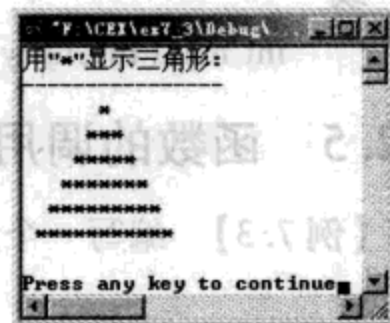


图 7.4 例 7.3 运行的结果


```

if(choice == 0)
    break;
switch(choice)
{
    case 1 : AreaOfRectangular(); break; //长方形
    case 2 : AreaOfTriangle(); break; //三角形
    case 3 : AreaOfCircle(); break; //圆形
    default : printf("输入有误,请在 0~4 之间选择.\n");
}

```

```
while(1);
```

```
void AreaOfRectangular()
```

```

{
    int x,y;
    printf("请输入长方形的长:");
    scanf("%d",&x);
    printf("请输入长方形的宽:");
    scanf("%d",&y);
    printf("面积为:%d\n",(x*y));
}

```

```
void AreaOfTriangle()
```

```

{
    int x,y;
    printf("请输入三角形的底:");
    scanf("%d",&x);
    printf("请输入三角形的高:");
    scanf("%d",&y);
    printf("面积为:%f\n",(x*y)/2);
}

```

```
void AreaOfCircle()
```

```

{
    int r;
    printf("请输入圆形的半径:");
    scanf("%d",&r);
    printf("面积为:%f\n",3.14*r*r);
}

```

程序运行结果如图 7.5 所示。

【注意点】

从例 7.4 看出,程序可以反复求出各个形状的面积,这是采用一个 do-while 循环实现的,请读者体会这种循环实现菜单的使用。

【例 7.5】 编程计算组合数 C_n^m 。设有两数 n, m , 且 $n > m$, 其组合数为 $\frac{n!}{m! \times (n-m)!}$ 。

【解题思路】

计算组合数, 首先要计算三个数 $n!, m!$, 及 $(n-m)!$, 为此需要编写三段求不同整数阶乘的代码。由于三段参考代码是类似的, 为简化程序设计, 可编写一个计算阶乘的函数, 然后用不同的参数三次调用该函数。

【参考代码】

```
#include <stdio.h>

float fun1(int);           //函数原型声明

void main()
{
    int n, m;
    float result;
    printf("请输入两个整数 n, m:");
    scanf("%d %d", &n, &m);
    result = fun1(n) / (fun1(m) * fun1(n-m));
    printf("所求组合数 = %f\n", result);
}

//计算 n 的阶乘

float fun1(int n)
{
    int i;
    float f = 1.0;
    for(i = 1; i <= n; i++)
        f = f * i;
    return f;               //通过 return 将运算结果 f 带回主函数
}
```

程序运行结果如图 7.6 所示。

7.3 函数的嵌套调用

C 语言中不允许作嵌套的函数定义。因此各函数之间是平行的, 不存在上一级函数和下一级函数的问题。但是 C 语言允许在一个函数的定义中出现对另一个函数的调用。这样就出现了函数的嵌套调用。即在被调函数中又调用其他函数。图 7.7 展示了这种关系, $\text{main}()$ 调用 $f1$, 在 $f1$ 中又调用函数 $f2$, $f2$ 又调用 $f3$, 在 $f3$ 完成后返回 $f2$ 的调用处, 继续 $f2$ 的执行, $f2$ 执行完之后返回 $f1$ 的调用处, $f1$ 又接着往下执行, 随后 $f1$ 又调用 $f4$ 函数, $f4$ 执行完后返回 $f1$, $f1$ 执行完返回 $\text{main}()$ 函数。 $\text{main}()$ 接着往下执行, $\text{main}()$ 完成后程序执行结束。

对应的程序结构如下:

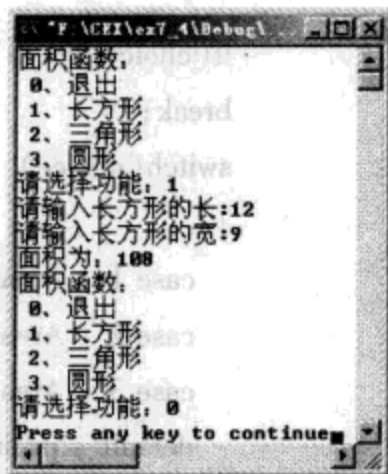


图 7.5 例 7.4 运行的结果

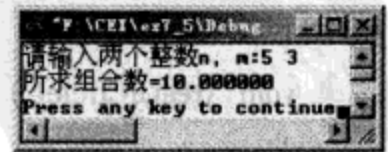


图 7.6 例 7.5 运行的结果

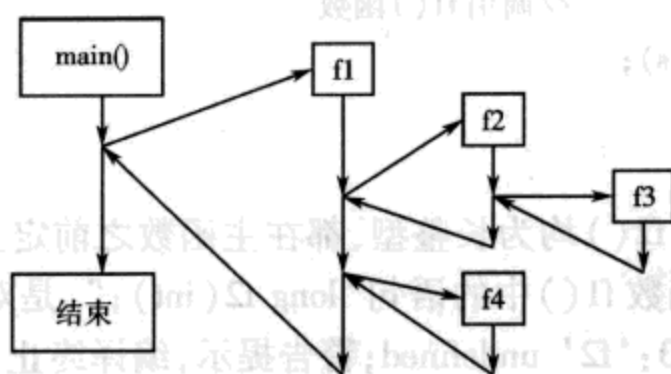


图 7.7 函数嵌套调用

函数类型说明 1 f1(形式参数类型序列 1);

函数类型说明 2 f2(形式参数类型序列 2);

函数类型说明 3 f3(形式参数类型序列 3);

函数类型说明 4 f4(形式参数类型序列 4);

【例 7.6】 计算 $s = 2^2! + 3^2!$

【解题思路】

本题可编写两个函数,一个是用来计算平方值的函数 f1,另一个是用来计算阶乘值的函数 f2。主函数先调 f1 计算出平方值,再在 f1 中以平方值为实参,调用 f2 计算其阶乘值,然后返回 f1,再返回主函数,在循环程序中计算累加和。

```
#include <stdio.h>
```

```
long f1(int p)
```

```
{
```

```
int k;
```

```
long r;
```

```
long f2(int);
```

```
k = p * p;
```

```
r = f2(k);
```

```
return r;
```

```
long f2(int q)
```

```
{
```

```
long f = 1;
```

```
int i;
```

```
for(i = 1; i <= q; i++)
```

```
f = f * i;
```

```
return f;
```

```
}
```

```
void main()
```

```
{
```

```
int i;
```

```
long s = 0;
```

```
for (i = 2; i <= 3; i++)
```

```
//在之前不需要进行函数原型说明
```

```
//因为 f1() 在主函数 main() 之前定义
```

```
//函数 f2() 的原型说明
```

```
//计算 p 的平方
```

```
//调用函数 f2()
```

```
//在之前不需要进行函数原型说明
```

```
//因为 f2() 在主函数 main() 之前定义
```

```

s = s + f1(i);           //调用 f1() 函数
printf("\ns = %ld\n", s);
}

```

【程序说明】

在程序中,函数 $f1()$ 和 $f2()$ 均为长整型,都在主函数之前定义,故不必再在主函数中对 $f1()$ 和 $f2()$ 加以说明。在函数 $f1()$ 中的语句“long $f2(int)$ ”,是对函数 $f2()$ 的原型说明,否则编译时出现“warning c4013: ‘f2’ undefined;警告提示,编译终止。在主程序中,执行循环程序依次把 i 值作为实参调用函数 $f1()$,求 i^2 值。在 $f1()$ 中又发生对函数 $f2()$ 的调用,这时是把 i^2 的值作为实参去调 $f2()$,在 $f2()$ 中完成求 $i^2!$ 的计算。 $f2()$ 执行完毕把 $i^2!$ 返回给 $f1()$,再由 $f1()$ 返回主函数实现累加。至此,由函数的嵌套调用实现了题目的要求。由于数值很大,所以函数和一些变量的类型都说明为长整型,否则会造成计算错误。

程序运行结果如图 7.8 所示。

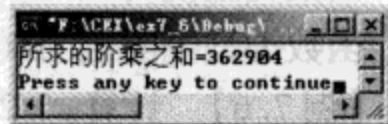


图 7.8 例 7.6 运行的结果

7.4 函数的递归调用

一个函数在它的函数体内调用它自身称为递归调用。这种函数称为递归函数。C 语言允许函数的递归调用。递归调用实际上是一种循环,这种调用的关键是要设置好循环条件,让循环进行到一定阶段时能一层层地退出,不能形成死循环,因此这一类程序的编写要特别细心。

例如,有函数 f 如下:

```

int f(int x)
{
    int y;
    z = f(y);
    return z;
}

```

这个函数是一个递归函数,但是运行该函数将无休止地调用其自身,这当然是不正确的。

【例 7.7】 用递归法计算 $n!$ 。

【解题思路】

在下列函数 $jc()$ 中,函数 $jc()$ 要调用函数本身,它是一递归函数。

```

long jc(int n)
{
    .....
    f = jc(n - 1) * n;
    .....
}

```

用递归法计算 $n!$,采用递归函数的正确描述为:

$$jc(n) = \begin{cases} n \times jc(n-1) & n > 1 \\ 1 & n = 1 \end{cases}$$

【参考代码】

```
#include <stdio.h>

long jc(int n)
{
    long f;
    if (n < 0) printf("n < 0, 输入数据有误!");
    else if (n == 0 || n == 1)
        f = 1;
    else
        f = jc(n - 1) * n;
    return(f);
}

void main()
{
    int n;
    long y;
    printf("输入一个正整数 n:");
    scanf("%d", &n);
    y = jc(n);
    printf("%d! = %ld\n", n, y);
}
```

【程序分析】

程序中给出的函数 `jc()` 是一个递归函数。主函数调用 `jc()` 后即执行函数 `jc()`，如果 $n < 0$ ， $n == 0$ 或 $n == 1$ 时都将结束函数的执行，否则就递归调用 `jc()` 函数自身。由于每次递归调用的实参为 $n - 1$ ，即把 $n - 1$ 的值赋予形参 n ，最后当 $n - 1$ 的值为 1 时的这一次递归调用，形参 n 的值也为 1，将使递归终止，然后可逐层退回。

下面再举例说明该过程。设执行本程序时输入为 5，即求 $5!$ 。在主函数中的调用语句即为 `y = jc(5)`，进入 `jc()` 函数后，由于 $n = 5$ ，不等于 0 或 1，故应执行 `f = jc(n - 1) * n`，即 `f = jc(5 - 1) * 5`。该语句对 `jc` 作递归调用即 `jc(4)`。进行四次递归后，`jc(1)` 的函数返回值为 1，`jc(2)` 的返回值为 $1 * 2 = 2$ ，`jc(3)` 的返回值为 $2 * 3 = 6$ ，`jc(4)` 的返回值为 $6 * 4 = 24$ ，最后返回值 `jc(5)` 为 $24 * 5 = 120$ 。

例 7.7 也可以不用递归的方法来完成。如可以用递推法，即从 1 开始乘以 2，再乘以 3……直到 n 。求阶乘时，递推法比递归法更容易理解和实现，但是有些问题则只能用递归算法才能实现。

程序运行的结果如图 7.9 所示。

7.5 数组作为函数参数

数组可以作为函数的参数使用，进行数据传送。

数组用作函数参数有两种形式，一种是把数组元素（下标变量）作为实参使用；另一种是把数组名作为函数的形参和实参使用。

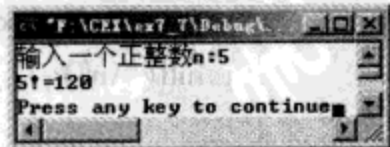


图 7.9 例 7.7 运行的结果

7.5.1 数组元素作函数实参

数组元素就是下标变量,它与普通变量并无区别。因此,它作为函数实参使用与普通变量是完全相同的,在发生函数调用时,把作为实参的数组元素的值传送给形参,实现单向的值传送。

【例 7.8】 输入某班级某门课程的 n 个成绩,使用子函数处理成绩,使成绩按五分制输出。

【解题思路】

用一个二维数组 `score[n][2]` 来存储序号(理解为学号)及成绩,用 `s(score[i][0], score[i][1])` 来处理第 i 个学生成绩等级。

【参考代码】

```
#include <stdio.h>
#define N 10
void s(int i,int score)
{
    if( score > 100 || score < 0)
        printf("第%d 学生成绩输入有误!\n",i+1);
    else if( score >= 90)
        printf("第%d 学生成绩等级为%8s\n",i+1,"优秀");
    else if( score >= 80)
        printf("第%d 学生成绩等级为%8s\n",i+1,"良好");
    else if( score >= 70)
        printf("第%d 学生成绩等级为%8s\n",i+1,"中等");
    else if( score >= 60)
        printf("第%d 学生成绩等级为%8s\n",i+1,"及格");
    else
        printf("第%d 学生成绩等级为%8s\n",i+1,"不及格");
}
void main()
{
    int i,score[N];
    printf("请输入%d 个学生成绩:\n",N);
    for(i=0;i<N;i++)
    {
        printf("第%d 个学生的成绩为:",i+1);
        scanf("%d",&score[i]);
        printf("\n");
    }
    printf("\n%d 个学生成绩等级为:\n",N);
    for(i=0;i<N;i++)
    {
        s(i,score[i]);
    }
}
```


程序运行结果如图 7.10 所示。

【注意点】

本程序中首先定义一个无返回值函数 $s()$ ，并说明其形参 i 和 $score$ 为整型变量。在函数体中根据 $score$ 值输出相应的结果。在 $main()$ 函数中用一个 for 语句输出数组各元素，每输出一个元素就以该元素作实参调用一次 $s()$ 函数，即把 i 及 $score[i]$ 的值传送给形参 i 和 $score$ ，供 $s()$ 函数使用。

7.5.2 数组名作为函数参数

用数组名作函数参数与用数组元素作实参有几点不同。

(1) 用数组元素作实参时，只要数组类型和函数的形参变量的类型一致，那么作为下标变量的数组元素的类型也应与函数形参变量的类型是一致的。因此，并不要求函数的形参也是下标变量。

用数组名作函数参数时，则要求形参与相对应的实参都必须是类型相同的数组，都必须有明确的数组说明。当形参与实参二者不一致时，即会发生错误。

(2) 在普通变量或下标变量作函数参数时，形参变量和实参变量是由编译系统分配的两个不同的内存单元。在函数调用时发生的值传送是把实参变量的值赋予形参变量。

在用数组名作函数参数时，不是进行值的传送，即不是把实参数组的每一个元素的值都赋予形参数组的各个元素。因为实际上形参数组并不存在，编译系统不为形参数组分配内存。那么，数据的传送是如何实现的呢？在前面曾介绍过，数组名就是数组的首地址。因此在数组名作函数参数时所进行的传送只是地址的传送，也就是说把实参数组的首地址赋予形参数组名。形参数组名取得该首地址之后，也就等于有了实在的数组。实际上是形参数组和实参数组为同一数组，共同拥有一段内存空间。

(3) 用变量作函数参数时，实参和形参之间是单向值传递，实参和形参是不同的单元；用数组名作函数参数时，实参和形参之间是单向地址传递，实参数组和形参数组共占用同一段内存单元。

【例 7.9】 输入某班级某门课程的 n 个成绩，使用函数处理成绩，要求成绩按五分制输出（用数组名作为参数）。

【解题思路】

用一个数组 $score[N]$ 来存储学生成绩，用 $s(int\ cj[])$ 来处理学生成绩等级。

假如函数 $s()$ 定义为：

```
void s(int cj[])
```

```
{
```

```
.....
```

```
}
```

数组名 $score$ 和 cj 实际上表示了数组的首地址，如图 7.11 所示。

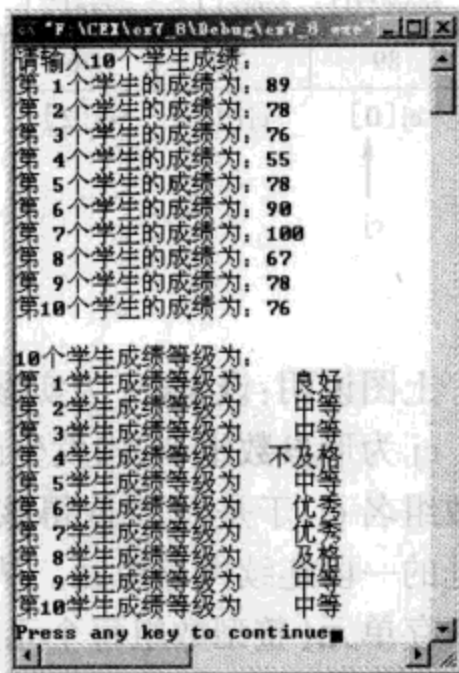


图 7.10 例 7.8 运行的结果

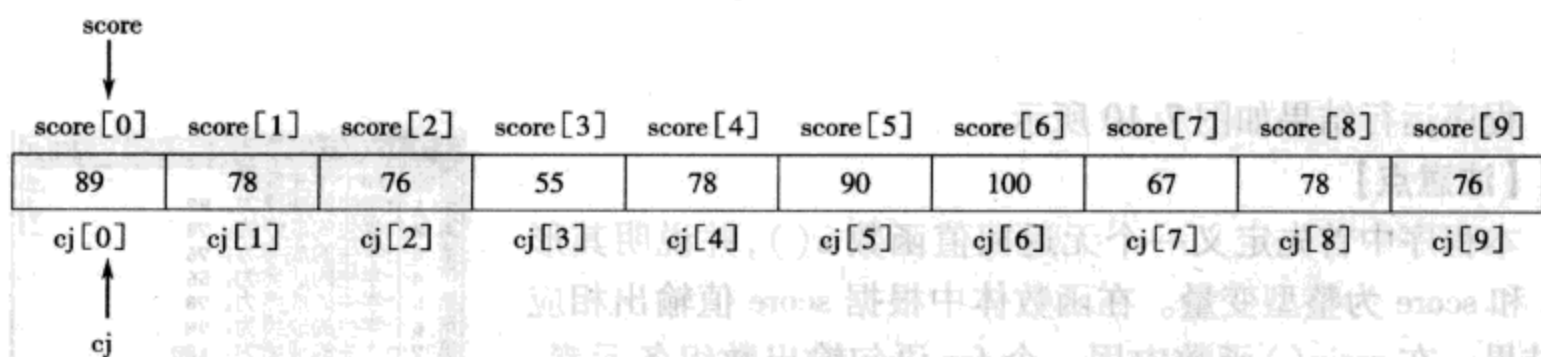


图 7.11 数组 score 与 cj 示意图

上图说明:设 score 为实参数组,类型为整型。假如其占有以 4000 为首地址的一块内存区。cj 为形参数组名,当发生函数调用时,进行地址传送,把实参数组 score 的首地址传送给形参数组名 cj,于是 cj 也取得该地址 4000。也就是说, score 和 cj 两数组共同占有以 4000 为首地址的一段连续内存单元。从图中还可以看出, score 和 cj 下标相同的元素实际上也占有相同的内存单元(整型数组每个元素占四个字节)。例如 score[0] 和 cj[0] 都占用 4000 到 4003 单元,所以 score[0] 等于 cj[0]。类推则有 score[i] 等于 cj[i]。

【参考代码】

```
#include <stdio.h>
```

```
#define N 10
```

```
void s(int cj[])
```

```
{
    int i;
```

```
    for(i=0;i<N;i++)
```

```
        if(cj[i]>100||cj[i]<0)
```

```
            printf("第%d 学生成绩输入有误!\n",i+1);
```

```
        else if(cj[i]>=90)
```

```
            printf("第%d 学生成绩等级为%8s\n",i+1,"优秀");
```

```
        else if(cj[i]>=80)
```

```
            printf("第%d 学生成绩等级为%8s\n",i+1,"良好");
```

```
        else if(cj[i]>=70)
```

```
            printf("第%d 学生成绩等级为%8s\n",i+1,"中等");
```

```
        else if(cj[i]>=60)
```

```
            printf("第%d 学生成绩等级为%8s\n",i+1,"及格");
```

```
        else
```

```
            printf("第%d 学生成绩等级为%8s\n",i+1,"不及格");
```

```
void main()
```

```
{
    int i,score[N];
```

```
    printf("请输入%d 个学生成绩:\n",N);
```

```
    for(i=0;i<N;i++)
```

```
    {
        printf("第%d 个学生的成绩为:",i+1);
```

```
scanf("%d",&score[i]);
}
printf("\n%d 个学生成绩等级为:\n",N);
s(score);
}
```

【注意点】

- (1) 符号常量 N 在子函数 s() 中照样可以使用。
- (2) 子函数 s() 的形参数组名 cj 也可以与主函数中的 score 同名, 它们都是指同一个内存单元的地址。

【例 7.10】在数组 score 中存放一个学生的 10 个成绩, 求平均成绩。

```
#include <stdio.h>
float average(int a[10])
{
    int i;
    int s = a[0];
    float aver;
    for(i = 1; i < 10; i++)
        s = s + a[i];
    aver = s/10;
    return aver;
}

void main()
{
    int score[10];
    float aver;
    int i;
    printf("请输入学生 10 个成绩:\n");
    for(i = 0; i < 10; i++)
        scanf("%d",&score[i]);
    aver = average(score);
    printf("平均成绩:%5.2f\n",aver);
}
```

程序运行结果如图 7.12 所示。

【程序说明】

本程序首先定义了一个实型函数 average, 有一个形参为整型数组 a, 长度为 10。在函数 average 中, 把各元素值相加求出平均值, 返回给主函数。主函数 main() 中首先完成数组 score 的输入, 然后以 score 作为实参调用 average() 函数, 函数返回值送 aver, 最后输出 aver 值。从运行情况可以看出, 程序实现了所要求的功能。

【注意点】

- (1) 形参数组和实参数组的类型必须一致, 否则将引起错误。

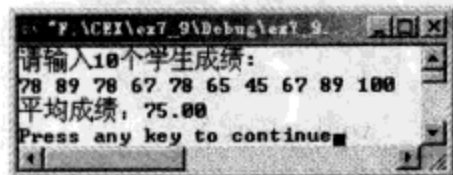


图 7.12 例 7.10 运行结果

(2) 形参数组和实参数组的长度可以不相同,因为在调用时,只传送首地址而不检查形参数组的长度。

7.6 变量的存储类别、作用域和生存期

引进函数概念后,由于变量可以在程序的不同位置说明、定义等原因(如在函数外定义变量和在函数内定义变量等不同),使得变量的概念被扩展。在 C 语言中,根据存储类别、作用域和生存期等不同,可将变量分成不同的类型。

(1) 按作用域不同,变量可分为局部变量和全局变量。

(2) 按存储类别,变量可分为自动变量(auto 型)、外部变量(extern 型)、寄存器变量(register 型)、静态变量(static 型)。

(3) 按生存期,变量可分为动态变量、静态变量。

7.6.1 局部变量与全局变量

在讨论函数的形参变量时曾经提到,形参变量只在被调用期间才分配内存单元,调用结束立即释放。这一点表明形参变量只有在函数内才是有效的,离开该函数就不能再使用了。这种变量有效性的范围称变量的作用域。

不仅形参变量,C 语言中所有的变量都有自己的作用域。变量说明的方式不同,其作用域也不同。C 语言中的变量,按作用域范围可分为两种,即局部变量和全局变量。

1. 局部变量

局部变量也称为内部变量。局部变量是在函数内作定义说明的。例如:

```
#include <stdio.h>
#define N 10                                //符号常量在整个程序中有效

float average( int a[ ])
{
    int i;
    int s = a[0];
    float aver;
    for( i = 1; i < N; i++ )
        s = s + a[i];
    aver = s/N;
    return aver;
}

void main()
{
    int score[N];
    float aver;
    int i;
    .....
}
```

//函数 average() 中 i、aver、s 有效

//主函数中 aver、i 有效

主函数与 average() 中变量 i、aver 虽然名称相同,但其作用范围不同。

在函数 `average()` 内定义了 `i`、`aver` 和 `s` 为一般变量。在 `average()` 的范围内 `i`、`aver` 和 `s` 有效,或者说 `i`、`aver` 和 `s` 变量的作用域限于 `average()` 内。同理 `main()` 中, `i` 和 `aver` 的作用域也只限于主函数内。

关于局部变量的作用域还要说明以下几点。

- (1) 主函数中定义的变量也只能在主函数中使用,不能在其他函数中使用。
- (2) 形参变量是属于被调函数的局部变量,实参变量是属于主调函数的局部变量。
- (3) 允许在不同的函数中使用相同的变量名,它们代表不同的对象,分配不同的单元,互不干扰,也不会发生混淆。
- (4) 在复合语句中也可定义变量,其作用域只在复合语句范围内。

例如:

```
main()
{
    int s,a;
    .....
    {
        int b;
        s = a + b;
    } //b 作用域
    .....
}
```

//s 和 a 作用域

【例 7.11】 局部变量的作用域。

```
#include <stdio.h>
void main()
{
    int a=1,b=3,c;
    c=a+b;
    {
        int c=10;
        printf("复合语句中的 c=%d\n",c);
    }
    printf("主函数中的 c=%d\n",c);
}
```

【程序说明】

本程序在 `main()` 中定义了 `a`、`b`、`c` 三个变量,其中 `c` 未赋初值。而在复合语句内又定义了一个变量 `c`,并赋初值为 10。应该注意这两个 `c` 不是同一个变量。在复合语句外,由 `main()` 定义的 `c` 起作用,而在复合语句内则由在复合语句内定义的 `c` 起作用。因此程序第 5 行的 `c` 为 `main()` 所定义,其值应为 4。第 8 行输出 `c` 值,该行在复合语句内,由复合语句内定义的 `c` 起作用,其初值为 10,故输出值为 10。第 10 行输出 `c` 值,`c` 是在整个程序中有效的,第 4 行对 `a` 赋值为 1, `b` 赋值 3, `c=a+b`, `c=4`。而第 10 行已在复合语句之外,输出的 `c` 应为 `main()` 所定义的 `c`,此 `c` 值由第 5 行已获得为 4,故输出也为 4。

程序运行结果如图 7.13 所示。

2. 全局变量

全局变量也称为外部变量,它是在函数外部定义的变量。全局变量作用范围大,但其作用范围与定义的位置有关。可大到整个程序,也可小到谁也不能使用(如定义在源文件末尾时,不用关键字 `extern`,谁也不能引用它)。

由于全局变量定义在函数外部,它不属于任何函数,一个全局变量只能定义一次,在编译时为其分配内存空间,其作用域为定义点到文件尾。

若全局变量和某个函数中的局部变量同名,则全局变量在函数中被屏蔽,该函数内访问的是局部变量,与同名的全局变量无任何关系。

例如:

```
int x,y;
void fun1()
{
    .....
}
float a,b;
int fun2()
{
    .....
}
main()
{
    int x,y;
    .....
}
```

//全局变量 x,y 作用域

//全局变量 a,b 作用域

//局部变量 x,y 作用域,
全局变量 x,y 被屏蔽

从上例可以看出,x、y、a、b 都是在函数外部定义的外部变量,都是全局变量,但主函数中的 x、y 却不能是全局变量,它仅是主函数中的局部变量而已。但 a、b 定义在函数 `fun1()` 之后,而在 `fun1()` 内又无对 a、b 的说明,所以它们在 `fun1()` 内无效。x、y 定义在源程序最前面,因此在 `fun1()`,`fun2()` 内不加说明也可使用,在 `main()` 中被屏蔽掉了。

【例 7.12】 全局变量作用域。

```
#include <stdio.h>
int add(int x,int y);
int z; //全局变量
void main()
{
    int z=15,x=2,y=3;
    printf("不调用 add 函数的结果:%d\n",x+y+z); //此时 z=15
    printf("调用 add 函数的结果:%4d\n",add(x,y)); //此时 z=5
}
int add(int x,int y)
```

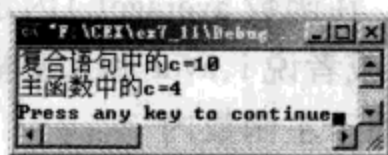


图 7.13 例 7.11 运行结果

```
z=5; //局部变量  
return (x+y+z); //z使用局部变量,此时 z=5
```

程序运行结果如图 7.14 所示。

【注意点】

同一个源文件中,外部变量 z 与局部变量 z 同名,则在函数 $\text{add}()$ 中的局部变量 z 的作用范围内,外部变量被“屏蔽”,即它不起作用,此时的 $z=5$ 。

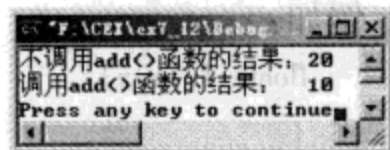


图 7.14 例 7.12 运行结果

7.6.2 变量的存储类别

1. 动态存储方式与静态存储方式

存储类别是指数据在内存中存储的位置。即明确了所说明的变量在内存中的存储位置,从而也明确了所说明的变量的作用域和生存期。

前面已经介绍过,从变量的作用域(即从空间)角度来分,可以分为全局变量和局部变量。从变量值存在的时间(即生存期)角度来分,可以分为静态存储方式和动态存储方式。

静态存储方式是指在程序运行期间分配固定的存储空间的方式。

动态存储方式是在程序运行期间根据需要进行动态的分配存储空间的方式。

用户存储空间可以分为三个部分:

- (1) 参考代码区;
- (2) 静态存储区;
- (3) 动态存储区(堆栈)。

数据主要存储在静态存储区和动态存储区。

静态存储区用来存放全局变量及静态类型的局部变量。全局变量全部存放在静态存储区,在程序开始执行时给全局变量分配存储区,程序执行完毕就释放。在程序执行过程中,它们占据固定的存储单元,而不动态地进行分配和释放。

动态存储区用来保存函数调用时的返回地址、自动类型的局部变量等。即动态存储区主要存放以下数据:

- (1) 函数形式参数;
- (2) 自动变量(未加 `static` 声明的局部变量);
- (3) 函数调用的现场保护和返回地址。

以上这些数据,在函数开始调用时分配动态存储空间,函数结束时释放这些空间。

在 C 语言中,每个变量和函数都有两个属性:数据类型和数据的存储类别。

2. auto 变量

`auto` 变量也称自动变量。当在函数内部或复合语句内定义变量时,如果没有指定存储类别或使用了 `auto` 说明符,系统就认为所定义的变量属于自动存储类别。

`auto` 变量的存储单元被分配在内存的动态存储区。每当进入函数体(或复合语句)时,系统自动为 `auto` 变量分配存储单元,退出时自动释放这些存储单元;当再次进入函数体(或复合语句)时,系统将为它们另行分配存储单元,变量的值不被保留。因此,这类局部变量的作用

域是从定义的位置起,到函数体(或复合语句)结束止。

由此,auto 变量具有这样的特点:动态存储区内为某个变量分配的存储单元位置随程序的运行而改变,变量中的初值也随之改变,所以这种变量必须赋初值,不同函数中使用了同名变量也不会相互影响。

例如:

```
float f(int x)          /* 定义 f 函数,x 为参数 */
```

```
{
```

```
    auto float y = 1.2, z = 2.4; /* 定义 y、z 自动变量 */
```

```
    x = y + z;
```

```
    .....
```

```
}
```

x 是形参,y,z 是自动变量,对 y 与 z 分别赋初值 1.2、2.4。执行完 f 函数后,自动释放 x、y、z 所占的存储单元。

3. register 变量

register 变量也称寄存器变量。寄存器变量也属于自动变量,它与 auto 变量的区别仅在于存储位置不同。用 register 变量说明的变量建议编译程序将变量的值保存在 CPU 的寄存器中,而不是在内存中,这样执行速度更快些。因此,可把频繁使用的少数变量指定为 register 变量,从而提高程序运行速度。

【例 7.13】求 $1! + 2! + 3! + \dots + 10!$ 。

```
#include <stdio.h>
```

```
int jc(int n)
```

```
{
```

```
    register int i, f = 1; /* 寄存器变量 */
```

```
    for(i = 1; i <= n; i++)
```

```
        f = f * i;
```

```
    return(f);
```

```
}
```

```
void main()
```

```
{
```

```
    int i, sum = 0;
```

```
    for(i = 1; i <= 10; i++)
```

```
        sum += jc(i);
```

```
    printf("1! + 2! + ... + 10! = %ld\n", sum);
```

```
}
```

程序运行结果如图 7.15 所示。

4. static 局部变量

当在函数体(或复合语句)内部用 static 来说明一个变量时,称该变量为静态局部变量。静态局部变量的作用域与 auto、register 类变量一样,但它与前者有两点本质上区别。

(1) 静态局部变量在整个程序运行期间,在内存的静态存储区中占据着永久性的存储单元。即使退出函数以后,下次再进入该函数时,静态局部变量仍使用原来的存储单元。由此可

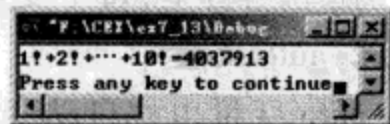


图 7.15 例 7.13 运行结果

见,静态局部变量的生存期一直延长到程序运行结束。
 例(2)静态局部变量的初值是在编译时赋给的,在程序执行期间不再赋初值,对未赋初值的静态局部变量,C编译系统自动赋初值为0(整型为0,float型为0.000000,char型为“\0”)。

【例 7.14】 考察静态局部变量的值。

```
#include <stdio.h>

int jc(int n)
{
    static int f = 1;          //静态变量的定义
    f = f * n;                  //静态变量的使用
    return (f);
}

void main()
{
    int n = 10, i, sum = 0;
    for(i = 1; i <= n; i++)
        sum += jc(i);
    printf("1! + 2! + ... + 10! = %ld\n", sum);
}
```

程序运行结果如图 7.15 所示。

【注意点】

请读者仔细阅读例 7.13 与例 7.14 的区别与联系,为什么例 7.13 中要用如下循环才能实现求某个数的阶乘:

```
for(i = 1; i <= n; i++)
    f = f * i;
```

而例 7.14 中只用: $f = f * n$; 即可求出呢?

原来例 7.14 中使用静态局部变量,当一次求 i 的阶乘后, f 中已经保存了 $i!$ 的值,下一次求 $(i+1)!$ 时,原来的 f 是静态变量,保存了上次的 f 值, $f = 1 \times 2 \times \dots \times i$, 因此执行 $f = f \times (i+1)$, 即求出了 $(i+1)!$ 。

5. 全局变量的存储类别

1) static 全局变量

当用 static 说明全局变量时,该变量称为“静态”全局变量。静态全局变量只限于本编译单位使用,不能被其他编译单位所引用。

2) extern 全局变量

在全局变量前加上 extern 的作用有以下两种。

(1) 在同一编译单位内用 extern 说明符来扩展全局变量的作用域。全局变量定义之后,当引用它的函数位置在前面时,应该在引用它的函数中用 extern 对此全局变量进行说明,以便通知编译程序:该变量是一个已在外部定义了的的全局变量,已经分配了存储单元,就不需要再为它另外开辟存储单元。

(2) 在不同编译单位内用 extern 说明符来扩展全局变量的作用域。一个 C 程序总是由许多函数组成的,这些函数可以分别存放在不同的源文件中,每个源文件可以单独编译。这些可

以单独编译的源文件称为“编译单位”。每一个程序由多个编译单位组成,并且在每个文件中定义所有的全局变量,其他用到这些全局变量的文件可用 `extern` 对这些变量进行说明,表示这些变量已在其他编译单位中定义,以通知编译系统不必再为它们开辟存储单元。

【例 7.15】 用 `extern` 声明外部变量,求 $2! + 3! + 4!$ 。

```
#include <stdio.h>

int jc(int n)
{
    int i, f = 1;
    for(i = 1; i <= n; i++)
        f = f * i;
    return(f);
}

void main()
{
    extern x, y, z;          //extern 声明全局变量
    printf("%d! + %d! + %d! = %d\n", x, y, z, jc(x) + jc(y) + jc(z));
}

int x = 2, y = 3, z = 4;    //全局变量
```

程序运行结果如图 7.16 所示。

【程序说明】

在本程序的最后一行定义了全局变量 `x`、`y`、`z`,但由于全局变量定义的位置在函数 `main()` 之后,因此,本来在 `main()` 函数中不能引用全局变量 `x`、`y`、`z`,现在在 `main()` 函数中用 `extern` 对 `x`、`y` 和 `z`

进行“全局变量声明”,就可以从“声明”处起,合法地使用该全局变量 `x`、`y` 和 `z`。

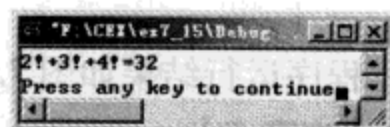


图 7.16 例 7.15 运行结果

* 7.7 内部函数与外部函数

函数本质上是全局的,因为一个函数要被另外的函数调用;但是,也可以指定函数不能被其他文件调用。根据函数能否被其他源文件调用,将函数区分为内部函数和外部函数。

7.7.1 内部函数

如果一个函数只能被本文件中其他函数调用,它称为内部函数。在定义内部函数时,在函数名和函数类型前面加 `static`。即

`static` 类型说明符 函数名(形参列表)

如:

```
static int fun(int a, int b)
```

内部函数又称静态函数。使用内部函数可以使函数只局限于所在文件;如果在不同的文件中有同名的内部函数,互不干扰。

7.7.2 外部函数

(1) 在定义函数时,如果在函数名最前端加上关键字 `extern`,则表示此函数是外部函数,可

供其他文件调用。如:

```
extern int fun(int a,int b)
```

如果在定义函数时省略了 `extern`, 则默认为外部函数。本书前面所用函数全为外部函数。

(2) 在需要调用此函数的文件中, 用 `extern` 声明所用的函数是外部函数。

7.8 案例应用举例

【例 7.15】“图书管理系统”中函数的使用。

【参考代码】(部分)

```
#include <stdio.h>

void bookadd() //图书增加
{
    .....
}

void booksearch() //图书查询
{
    .....
}

void bookupdate() //图书修改
{
    .....
}

void bookdelete() //图书删除
{
    .....
}

void cardadd() //借书卡增加
{
    .....
}

void cardsearch() //借书卡查询
{
    .....
}

void cardupdate() //借书卡修改
{
    .....
}

void carddelete() //借书卡删除
{
    .....
}

void bookborrow() //借书
{
    .....
}

void bookreturn() //还书
{
    .....
}

void bookborr() //借还书查询
{
    .....
}

void main()
{
    .....;bookadd();.....;
    .....;bookdelete();.....;
    .....;bookupdate();.....;
    .....;booksearch();.....;
    .....;cardadd();.....;
    .....;carddelete();.....;
    .....;cardupdate();.....;
    .....;cardsearch();.....;
    .....;bookborrow();.....;
```

```
.....;bookreturn();.....;
```

```
.....;bookborr();.....;
```

【程序说明】

“图书管理系统”共有 11 个子函数和一个主函数。部分参考代码前面已经介绍,这说明 C 语言程序是由函数组成的,这是 C 语言的一大特点,没有函数也就不成为 C 语言了。

本章小结

(1) 函数是 C 语言的基本组成部分,任何复杂的 C 语言程序都是由函数组成的。每个 C 程序有且仅有一个主函数,而且总是从主函数开始执行。可以将主函数放在整个程序的前面,也可以放在其他函数的后面。

(2) 函数是 C 语言中重要的概念,也是程序设计的重要手段。本章重点介绍了函数的定义与调用、函数的参数传递、函数的嵌套与递归、变量的作用域和存储类型等。

习题七

一、选择题

1. 以下说法中正确的是()。

- A) C 语言程序总是从第一个定义的函数开始执行
- B) 在 C 语言程序中,要调用的函数必须在 `main()` 函数中定义
- C) C 语言程序总是从 `main()` 函数开始执行
- D) C 语言程序中的 `main()` 函数必须放在程序的开始部分

2. 有以下程序:

```
#include <stdio.h>
```

```
float fun(int x,int y)
```

```
{
```

```
    return(x + y);
```

```
}
```

```
void main()
```

```
{
```

```
    int a = 2, b = 5, c = 8;
```

```
    printf("%3.0f\n", fun((int)fun(a + c, b), a - c));
```

```
}
```

程序运行后的输出结果是()。

A) 编译出错

B) 9

C) 21

D) 9.0

3. 有以下程序:

```
#include <stdio.h>
```

```
int fun(int a,int b)
```

```

{
    if(b==0) return a;
    else return (fun( --a, --b));
}

```

```

void main()
{ printf("%d\n", fun(4,2)); }

```

程序的运行结果是()。

- A)1 B)2 C)3 D)4

4. 下面的程序段运行后,输出结果是()。

```

#include <stdio.h>
void main()
{
    int i,j,x=0;
    static int a[8][8];
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    a[i][j]=2*i+j;
    for(i=0;i<8;i++)
    x+=a[i][j];
    printf("%d\n",x);
}

```

- A)9 B)不确定值 C)0 D)18

5. 以下程序的输出结果是()。

```

#include <stdio.h>
func(int a,int b)
{
    int c;
    c=a+b;
    return c;
}
void main()
{
    int x=3,y=4,z=5,r;
    r=func((x--,y++,x+y),z--);
    printf("%d\n",r);
}

```

- A)11 B)12 C)23 D)21

6. 下列程序执行后的输出结果是()。

```

#include <stdio.h>

```

```
void func(int a, int b[])
```

```
{
    b[0] = a + 6;
}
```

```
void main()
```

```
{
    int a = 5, b[5];
    b[0] = 3;
    func(a, b);
    printf("%d\n", b[0]);
}
```

A) 11

B) 12

C) 13

D) 14

7. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
int f(int a, int b)
```

```
{
    int c;
    c = a;
    if(a > b)
        c = 1;
    else if(a == b)
        c = 0;
    else c = -1;
    return(c);
}
```

```
void main()
```

```
{
    int i = 2, p;
    p = f(i, i + 1);
    printf("%d\n", p);
}
```

A) -1

B) 0

C) 1

D) 2

8. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
void reverse(int a[], int n)
```

```
{
    int i, t;
    for(i = 0; i < n/2; i++)
    {

```

```
if(b == 0) return a;
```

```
else return (fun(--a, --b));
```

```
void main()
```

```
{ printf("%d\n", fun(4, 5)); }
```

() 是果然行运的序

(A) 1 (B) 2

4. 下面的程序运行后, 输出结果是

```
#include <stdio.h>
```

```
void main()
```

```
int i, x = 0;
```

```
static int a[8];
```

```
for(i = 0; i < 8; i++)
```

```
for(j = 0; j < 3; j++)
```

```
a[i][j] = 2 * i + j;
```

```
for(i = 0; i < 8; i++)
```

```
for(j = 0; j < 3; j++)
```

```
printf("%d\n", x);
```

(B) 不确定

(A) 0

2. 以下程序的输出结果是

```
#include <stdio.h>
```

```
func(int a, int b)
```

```
int c;
```

```
if(a > b)
```

```
return a;
```

```
else return b;
```

```
void main()
```

```
{ printf("%d\n", func(4, 5)); }
```

```
int x = 3, y = 4, z = 5;
```

```
z = func(x, y);
```

```
printf("%d\n", z);
```

```
void main()
```

```
{ printf("%d\n", z); }
```

6. 下列程序执行的输出结果是

```
#include <stdio.h>
```

```

    t = a[i];
    a[i] = a[n-1-i];
    a[n-1-i] = t;
}
}

```

```

void main()

```

```

{
    int b[10] = {1,2,3,4,5,6,7,8,9,10};
    int i,s=0;
    reverse(b,8);
    for(i=6;i<10;i++)
        s += b[i];
    printf(" %d\n",s);
}

```

A)22

B)10

C)34

D)30

9. 以下程序的输出结果是()。

```

#include <stdio.h>

```

```

double f(int n)
{
    int i; double s;
    s = 1.0;
    for(i=1;i<=n;i++)
        s += 1.0/i;
    return s;
}

void main()

```

```

{
    int i,m=3;
    double a=0.0;
    for(i=0;i<m;i++)
        a += f(i);
    printf("%f\n",a);
}

```

A)5.500000

B)3.000000

C)4.000000

D)8.25

10. 以下程序中函数 sort 的功能是对 a 数组中的数据进行由大到小的排序,程序运行后的输出结果是()。

```

#include <stdio.h>

```

```

void sort(int a[],int n)

```



```

{
    int i,j,t;
    for(i=0;i<n-1;i++)
    for(j=i+1;j<n;j++)
    if(a[i]<a[j])
    {
        t=a[i];a[i]=a[j];a[j]=t;
    }
}

```

void main()

```

{
    int aa[10] = {1,2,3,4,5,6,7,8,9,10},i;
    sort(&aa[3],5);
    for(i=0;i<10;i++)
    printf("%d",aa[i]);
    printf("\n");
}

```

A) 1,2,3,4,5,6,7,8,9,10,

B) 10,9,8,7,6,5,4,3,2,1,

C) 1,2,3,8,7,6,5,4,9,10,

D) 1,2,10,9,8,7,6,5,4,3,

11. 请阅读以下程序,其运行后的输出是()。

```
#include <stdio.h>
```

```
void fun(int s[])
```

```

{
    static int j=0;
    do
    {
        s[j] += s[j+1];
    }
    while(++j<2);
}

```

main()

```

{
    int k,a[10] = {1,2,3,4,5};
    for(k=1;k<3;k++)
    fun(a);
    for(k=0;k<5;k++)
    printf("%d",a[k]);
}

```

A) 34756

B) 23445

C) 35745

(D) 12345

12. 有以下函数定义:

```
int mypr (double a, double b)
{
    return a * b;
}
```

若下面选项中所用变量都已正确定义并赋值,错误的函数调用是()。

A) if(mypr(x,y)) { }

B) z = mypr(mypr(x,y) , mypr(x,y));

C) z = mypr(mypr(x,y) x,y);

D) mypr(x,y);

13. 有以下程序:

```
#include <stdio.h>
int f( int n)
{
    if( n == 1)
        return 1;
    else
        return f( n - 1 ) + 1;
}
void main( )
{
    int i, j = 0;
    for( i = 1; i < 3; i ++ )
        j + = f( i );
    printf( "%d\n", j );
}
```

程序运行后的输出结果是()。

A) 4

B) 3

C) 2

D) 1

14. 在 C 语言中,变量的默认存储类别是()。

A) auto

B) static

C) extern

D) 无存储类别

15. 有以下程序:

```
#include <stdio.h>
int f1( int x, int y)
{
    return x > y ? x : y;
}
```

```
int f2( int x, int y)
{
    return x > y ? y : x;
}
```

```
void main( )
{
    int a = 4, b = 3, c = 5, d = 2, e, f, g;
    e = f2( f1( a, b ), f1( c, d ) );
```

```
f = f1(f2(a,b),f2(c,d));
g = a + b + c + d - e - f;
printf("%d,%d,%d\n",e,f,g);
```

程序运行后的输出结果是()。

- A) 4,3,7 B) 3,4,7 C) 5,2,7 D) 2,5,7

16. 下列函数的运行结果是()。

```
#include <stdio.h>
void main()
{ int i=2,p;
  int j,k;
  j=i;
  k=++i;
  p=f(j,k);
  printf("%d",p);
}
int f(int a,int b)
{ int c;
  if(a>b)
    c=1;
  else if(a==b)
    c=0;
  else c=-1;
  return(c);
}
```

- A) -1 B) 1 C) 2 D) 编译出错,无法运行

17. 以下叙述中错误的是()。

- A) 函数中的自动变量可以赋初值,每调用一次,赋一次初值
B) 在调用函数时,实参和对应形参在类型上只需赋值兼容
C) 全局变量的隐含类别是自动存储类别
D) 函数形参可以说明为寄存器变量

18. 以下叙述错误的是()。

- A) 在函数之外定义的变量称全局变量,全局变量可以被程序中的所有函数调用
B) 全局变量定义和全局变量的说明的含义不同
C) 在一个函数中既可以使用本函数中的局部变量,也可以使用全局变量
D) 若在同一个源文件中,全局变量与局部变量同名,则在局部变量的作用范围内,全局变量不起作用

19. 下面的函数调用语句中 func() 函数的实参个数是()。

```
func(f2(v1,v2),(v3,v4,v5),(v6,max(v7,v8)));
```

A)3

B)4

C)5

D)6

二、填空题

1. 以下程序的输出结果是_____。

```
#include <stdio.h>
unsigned fun(unsigned num)
{
    unsigned k = 1;
    do
    {
        k * = num % 10;
        num /= 10;
    }
    while(num);
    return k;
}
void main()
{
    unsigned n = 12;
    printf("%d\n", fun(n));
}
```

2. 下列程序执行后输出的结果是_____。

```
#include <stdio.h>
int f(int a)
{
    int b = 0;
    static c = 3;
    a = c ++ , b ++ ;
    return(a);
}
void main()
{
    int a = 2, i, k;
    for(i = 0; i < 2; i ++ )
        k = f(a ++ );
    printf("%d\n", k);
}
```

3. 以下程序的输出结果是_____。

```
#include <stdio.h>
```

```

double sub( double x, double y, double z)
{
    y = 1.0;
    z = z + x;
    return z;
}

void main()
{
    double a = 2.5, b = 9.0;
    printf( "%f\n", sub( b - a, a, a) );
}

```

4. 以下程序的功能是:删去一维数组中所有相同的数,使之只剩一个。数组中的数已按由小到大的顺序排列,函数返回删除后数组中数据的个数。

例如,若一维数组中的数据是:

2 2 2 3 4 4 5 6 6 6 6 7 7 8 9 9 10 10 10

删除后,数组中的内容应该是:

2 3 4 5 6 7 8 9 10。

请填空。

```

#include <stdio.h>
#define N 80
int fun( int a[ ], int n)
{
    int i, j = 1;
    for( i = 1; i < n; i++)
        if( a[j-1] _____ a[i] )
            a[j++] = a[i];
    return _____;
}

void main()
{
    int a[N] = { 2, 2, 2, 3, 4, 4, 5, 6, 6, 6, 6, 7, 7, 8, 9, 9, 10, 10, 10 };
    printf( "原数据序列为:\n" );
    for( i = 0; i < n; i++)
        printf( "%3d", a[i] );
    printf( "\n" );
    n = fun( a, n );
    printf( "删除后的数据序列:\n" );
    for( i = 0; i < n; i++)
        printf( "%3d", a[i] );
}

```



```
printf("\n");
}
```

5. 以下程序的输出结果是_____。

```
#include <stdio.h>
```

```
fun1(int a,int b)
```

```
{
    int c;
```

```
    a += a;
```

```
    b += b;
```

```
    c = fun2(a,b);
```

```
    return c * c;
}
```

```
fun2(int a,int b)
```

```
{
```

```
    int c;
```

```
    c = a * b % 3;
```

```
    return c;
}
```

```
void main()
```

```
{
```

```
    int x = 11, y = 19;
```

```
    printf("%d\n", fun1(x,y));
}
```

6. 下面函数的功能是将一个字符串的内容颠倒过来,请填空。

```
#include <stdio.h>
```

```
void fun(char str[])
```

```
{
```

```
    int i, j, k;
```

```
    for(i = 0, j = _____; i < j; i++, j--)
```

```
    {
```

```
        k = str[i];
```

```
        str[i] = str[j];
```

```
        str[j] = k;
    }
```

```
}
```

7. 以下函数用以求 x 的 y 次方,请填空。

```
double fun(double x,int y)
```

```
{
```

```
    int i;
```

```

double z = 1;
for(i = 1; _____; i++)
z = _____;
return z;
}

```

8. 以下函数 rotate 的功能是:将 a 所指 N 行 N 列的二维数组中的最后一行放到 b 所指二维数组的第 0 列中,把 a 所指二维数组中的第 0 行放到 b 所指二维数组的最后一列中,b 所指二维数组中其他数据不变。

```

#include <stdio.h>
#define N 4
void rotate(int a[][N],int b[][N])
{
    int i,j;
    for(i=0;i<N;i++)
    {
        b[i][N-1] = _____;
        _____ = a[N-1][i];
    }
}

```

9. 阅读下面程序,则程序的执行结果为 _____。

```

#include <stdio.h>
int fun(int a,int b)
{
    int z;
    z = a/b;
    return z;
}

```

```

void main()
{

```

```

    int a = 30, b = 20, z;
    z = fun(a + b, a - b);
    printf("%d\n", z);
}

```

10. 下述程序的输出结果是_____。

```

#include <stdio.h>

```

```

int fun(int x)
{

```

```

    int p;
    if(x == 0 || x == 1)

```

```

return 3;
else
p = x - fun(x - 2);
return p;
}

void main()
{

```

```

printf("%d\n", fun(9));
}

```

11. 下面程序的输出结果是_____。

```

#include <stdio.h>
long fun(int n)
{

```

```

long s;

```

```

if((n == 1) || (n == 2))

```

```

s = 2;

```

```

else

```

```

s = n + fun(n - 1);

```

```

return(s);
}

```

```

void main()
{

```

```

long x;

```

```

x = fun(4);

```

```

printf("%ld\n", x);
}

```

12. 本程序用改进的冒泡法对数组 a[n] 的元素从小到大排序, 请在程序空白处填空。

```

#include <stdio.h>

```

```

void bubble(int a[], int n)
{

```

```

int j, k, jmax, temp;

```

```

jmax = _____;

```

```

do
{

```

```

k = 0;

```

```

for(j = 0; j < jmax; j++)

```

```

if(a[j] > a[j + 1])
{

```

```

temp = a[j]; a[j] = a[j + 1]; a[j + 1] = temp;

```

```

        k = _____;
    }
    jmax = k;
    while(jmax > 0);
}

void main()
{
    int i, a[10] = {7, 43, 3, 2, 4, 6, 8, 22, 11, 34};
    bubble(a, 10);
    for(i = 0; i < 10; i++)
        printf("%3d ", a[i]);
    printf("\n");
}

```

三、程序调试和编程题

1. 编写程序, 其中含有函数 `int mymod(int a, int b)` 用以求 `a` 被 `b` 除之后的余数。
2. 编写程序求: $1 - \frac{1}{2} + \frac{1}{3} - \dots + (-1)^{n-1} \frac{1}{n}$ 。
3. 编写函数, 连接两个字符串。
4. 编写函数, 根据整型形参 `m` 的值, 计算如下公式的值:

$$1 - \frac{1}{2 * 2} - \frac{1}{3 * 3} - \dots - \frac{1}{m * m}$$

若 $m = 5$, 则应输出 0.536389。

5. 编写函数, 用以求表达式 $x * x - 5 * x + 4$, `x` 作为参数传送给函数, 调用此函数求:

$$y1 = 2 * 2 - 5 * 2 + 4$$

$$y2 = (x + 15) * (x + 15) - 5 * (x + 15) + 4$$

$$y3 = \sin x * \sin x - 5 * \sin x + 4$$

PDF

第8章 指针

指针是C语言特点之一,也是C语言的精华,极大地丰富了C语言的功能。它是C语言中广泛使用的一种数据类型。利用指针可以直接访问内存单元中的数据、表示复杂的数据结构、动态分配内存、方便地使用字符串和数组等,使程序更加简洁、紧凑和高效。学习指针是学习C语言中最重要的一环,能否正确理解和使用指针是是否掌握C语言的一个标志。不过需要指出的是,大多数问题不用指针也可以解决,只不过在C语言中,有些问题用指针可以解决得更好。

8.1 指针的概念和定义

8.1.1 地址和指针的概念

1. 地址

在过去的编程中定义或说明变量后,编译系统就已经为定义的变量分配相应的内存单元。也就是说,每个变量在内存中会有固定的位置,即具有内存单元的“门牌号码”。由于变量的数据类型不同,它所占用的内存字节数也有所不同。在C语言中,经常要引用变量的地址,若一变量类型是基本的数据类型(如整型、浮点型、字符型等),则只需要在变量名的前面加上取地址运算符“&”,假设*i*是int型,*f*是float型,*c*是char型,则&*i*、&*f*、&*c*就都是变量的地址。

若变量类型是一维数组,如:

```
int a[5];
```

数组*a*的首地址是*a*,而不能是&*a*。在C语言中单独出现的数组名,代表数组的首地址。数组通常占据一段连续的内存空间。C语言规定,数组的地址就是数组中第一个元素的地址,也称首地址。因此上面*a*和&*a*[0]是一样的。

2. 指针

简单地说,指针就是地址。地址是计算机内存管理中的重要概念。计算机内存中指令的存取均是通过地址来进行的。在程序中对变量的访问是通过变量名进行的,变量名和变量的地址是一一对应的,访问变量的地址采用“按名存取”,也就是说系统是通过变量的地址来访问变量的。

变量与内存中某单元对应,而变量的值是变量所在内存单元的内容。对于一个内存单元来说,单元的地址即为指针,其中存放的数据才是该单元的内容。在C语言中,允许用一个变量来存放指针,这种变量称为指针变量。因此,一个指针变量的值就是某个内存单元的地址或称为某内存单元的指针,也可称为地址的地址。

如何真正理解指针呢?

例如,定义一个整型变量*i*,编译系统会自动为整型变量*i*分配4个字节的内存空间(在Visual C++6.0中),假设这4个字节的地址编号是2000、2001、2002、2003;当用“scanf(“%d”,

&i);”语句从键盘中输入 5 时,系统进行的操作是把 5 存入 2000、2001、2002、2003 地址单元中;当用“printf(“%d”,i);”语句输出 i 值时,系统进行的操作是把 2000、2001、2002、2003 地址单元的值输出。实际上整个输入/输出过程中只是直接对变量的地址单元进行操作。这种直接引用变量的方式称为直接访问方式。

还有一种访问方式叫间接访问方式,将变量 i 的地址(所在数据单元的首地址)2000 存放到另一个变量 p 中,通过这个变量 p 中存放的地址 2000 来访问指定变量 i 的值。存放变量 i 的地址的变量 p 称为指针变量,通常简称为指针。图 8.1(a)所示,可简化为图 8.1(b)所示。

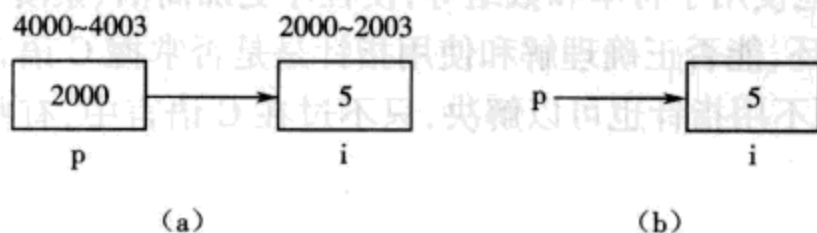


图 8.1 指针与变量

(a) 内存地址表示; (b) 指针表示简图

指针变量的说明如下。

- (1) 指针变量存放的只能是地址,但不关心具体的地址是什么。
- (2) 指针变量本身也是变量,也要占用存储单元。

在 Visual C++ 6.0 中指针变量占用 4 个字节的空间。例如:

```
#include <stdio.h>
void main()
{
    int * p, i = 3;
    p = &i;
    printf("p 占用的字节数:%d\n", sizeof(p));
    printf("i 占用的字节数:%d\n", sizeof(i));
}
```

其程序运行结果如图 8.2 所示。

再强调一下,指针变量也是变量,是存放地址的一个变量。指针是地址,指针变量的值是地址。所以说,地址就是指针,指针就是地址。

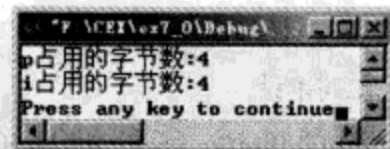


图 8.2 指针占用字节数

8.1.2 指针变量的定义

指针变量也是变量,与其他类型的变量一样,使用前必须先定义。

指针定义的一般形式为:

类型说明符 * 指针变量名 1, * 指针变量名 2, ……;

说明:

- (1) 这里所说的类型说明符是指,定义的指针变量可以指向何种数据类型的变量,或者说定义的指针变量中可以存放何种数据类型变量的起始地址。
- (2) * 是一个说明符,说明跟在它后面的标识符是指针变量,只能存放地址。
- (3) 一行中可以定义多个指针变量,各变量之间用逗号分开,它们将属于同一类型的

变量。

例如：

```
int *p;           //说明 p 是指向 int 类型变量的指针变量
char *s;          //说明 s 是指向 char 类型变量的指针变量
double *d;        //说明 d 是指向 double 类型变量的指针变量
float *f1, *f2;    //说明 f1 和 f2 是指向 float 类型变量的指针变量
```

为了叙述方便,根据 8.1.2 节中指针变量的概念,以后文中有时使用的“指针”就理解为指针变量,请读者注意。

8.1.3 指针的操作

1. 指针变量的引用

指针变量同普通变量一样,使用之前不仅要定义说明,而且必须赋予具体的值,未经赋值的指针变量不能使用。指针变量的赋值只能赋予地址。

下面是两个有关的运算符。

(1)&: 取地址运算符,表示取变量的地址。

其一般形式为:

& 变量名

如 &a 表示变量 a 的地址,&b 表示变量 b 的地址。变量本身必须预先说明。

(2)*: 指针运算符(或称“间接访问”运算符),表示访问指针变量指向的变量的值。

表 8.1 列出了对变量 i 和指针变量 p 进行引用的常用方法(参照图 8.3 所示的定义)。

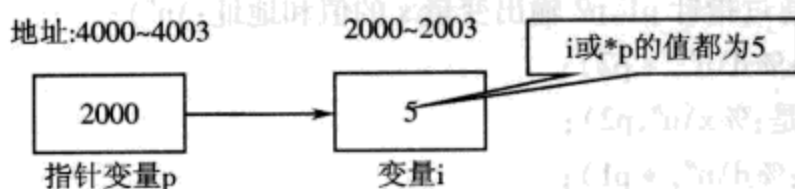


图 8.3 直接引用与间接引用

表 8.1 & 与 * 运算符

&i	表示变量 i 的地址; &i = 2000
p	指针变量 p 的值; p = 2000 = &i
*p	表示指针变量 p 指向的变量的值: *p = i = 5
*&i	相当于 *(&i), *(&i) = *p = 5
&*p	相当于 &(*p), &(*p) = &i = p = 2000
&p	表示指针变量 p 的地址, &p = 4000
*&p	相当于 *(&p), *(&p) = p = 2000

从表中可以看出两个运算符的结合性是“自右至左”。“&”和“*”是一种相反的运算,如 * &p = &*p = p, 但 * &i 和 &*i 不相等, 因为“*”后面是指针变量。

【例 8.1】通过指针直接或间接求变量的值和地址。

```
#include <stdio.h>
```

```
void main()
```

```

int x = 12, y = 24;
int * p1, * p2;

//第1步
p1 = &x;
printf("p1 = &x 通过指针 p1 输出变量 x 的值和 x 的地址:\n");
printf("x 的值是:%d\n", * p1);
printf("x 的地址是:%x\n", p1);
p2 = &y;
printf("p2 = &y;通过指针 p2 输出变量 y 的值和 y 的地址:\n");
printf("y 的值是:%d\n", * p2);
printf("y 的地址是:%x\n", p2);

/* 通过指针 p2 间接访问变量 x 和 y 的值 */
//第2步

* p2 = * p1;
printf("* p2 = * p1;通过指针 p2 输出变量 y 的值和地址:\n");
printf("y 的值是:%d\n", * p2);
printf("y 的地址是:%x\n", p2);

/* 改变指针 p2 的指向 */
//第3步

p2 = p1;
printf("p2 = p1;通过指针 p1, p2 输出变量 x 的值和地址:\n");
printf("x 的值是:%d\n", * p2);
printf("x 的地址是:%x\n", p2);
printf("x 的值是:%d\n", * p1);
printf("x 的地址是:%x\n", p1);

```

【程序说明】

程序执行第 1、2、3 步后,各指针变量与变量的值如图 8.4 所示。程序运行结果如图 8.5 所示。

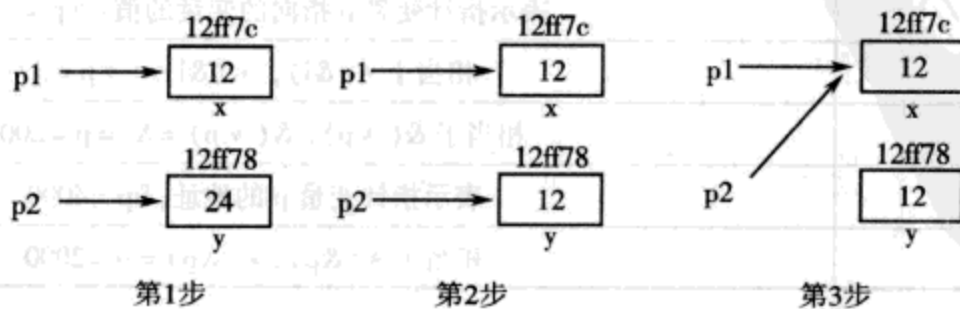


图 8.4 例 8.1 的指针操作

2. 指针变量的初始化

设有指向整型变量的指针变量 p,如要把整型变量 a 的地址赋予 p 可以有以下两种方式。

1) 指针变量初始化的方法

```
int a;
```

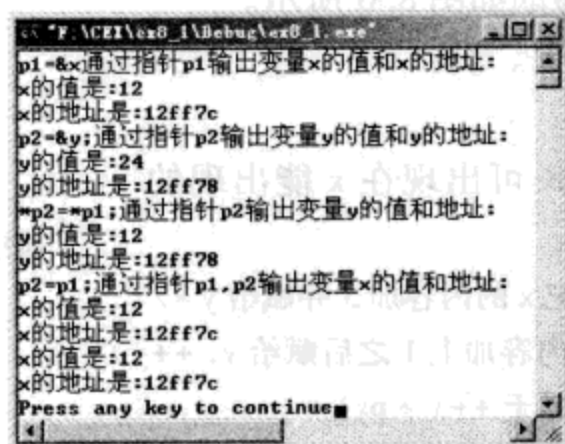


图 8.5 例 8.1 执行结果

```
int *p = &a;
```

2) 赋值语句的方法

```
int a;
int *p;
p = &a;
```

不允许把一个数赋予指针变量,故下面的赋值是错误的:

```
int *p;
p = 1000;
```

被赋值的指针变量前不能再加“*”说明符,如写为 *p = &a 也是错误的。不允许对没有初始化的指针变量作如下操作: *ip = i

例如:

```
#include <stdio.h>
void main()
{
    int i = 200;
    int *ip;
    *ip = i;
    printf("ip 指向的变量的值:%d\n", *ip);
}
```

程序运行会报错:“local variable 'ip' used without having been initialized”。

3. 指针变量的移动

指针变量的移动有三种操作。

- (1) p = p + n (或 p = p - n);
- (2) p ++;
- (3) p --。

其中, n 是一个整数,也可以是一个整型表达式。(1)中表达式的作用是使指针 p 向前(或向后)移动 n 个单位长度; p ++ (p --) 的作用是使 p 向前(或向后)移动 1 个单位长度。

例如:

```
int i = 3, *p = &i;
p ++;
p = p + 2;
```

执行完以上程序段后,指针的移动如图 8.6 所示。

指针变量可出现在表达式中。设:

```
int x, y, *px = &x;
```

指针变量 px 指向整数 x , 则 $*px$ 可出现在 x 能出现的任何地方。例如:

```
y = *px + 5;    /* 表示把 x 的内容加 5 并赋给 y */
```

```
y = ++ *px;    /* px 的内容加上 1 之后赋给 y, ++
```

```
    *px 相当于 ++(*px) */
```

```
y = *px ++;    /* 相当于 y = *px; px ++; */
```

【例 8.2】输入 a 和 b 两个整数,按先大后小的顺序输出 a 和 b 。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int *p1, *p2, *p, a, b;
```

```
    printf("请输入 a, b 的值:");
```

```
    scanf("%d %d", &a, &b);
```

```
    p1 = &a;
```

```
    p2 = &b;
```

```
    if(a < b)
```

```
    { p = p1; p1 = p2; p2 = p; }
```

```
    printf("a = %d, b = %d\n", a, b);
```

```
    printf("最大值 = %d, 最小值 = %d\n", *p1, *p2);
```

```
}
```

程序运行结果如图 8.7 所示。

8.1.4 指向指针的指针

前面介绍的指针变量都是指向普通变量的指针。从存储角度来考虑,指向普通变量的指针变量存放的是普通变量的地址。

若某指针变量中存放的不是普通变量的地址,而是另一个指针变量的地址,则该指针变量便被称为指向指针变量的指针变量,可以简称为“指向指针的指针”。

如果指针变量存放的是数组的地址或函数的地址,则该指针变量便被称为指向“数组”的指针变量或指向“函数”的指针变量。这将在后面的有关章节中进行详细介绍。

指向指针变量的指针变量的定义形式如下:

类型说明符 ****** 指针变量名 [= 初值];

定义指向指针变量的指针变量时应注意以下几点。

(1) 指向指针变量的指针变量定义时前面必须有“******”号,指向的变量的数据类型仍由“类型说明符”确定。

(2) 在定义的同时可以赋初值。初值必须是一个指针变量的地址,通常形式为“& 指针变量名”。例如:



图 8.6 指针的移动

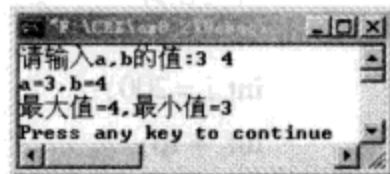


图 8.7 例 8.2 运行结果


```
int i = 12;
int * p = &i;
int ** q = &p;
```

此程序段定义了一个整型变量 i 、一个指针变量 p 、一个指向指针变量的指针变量 q 。三者之间关系如图 8.8 所示。

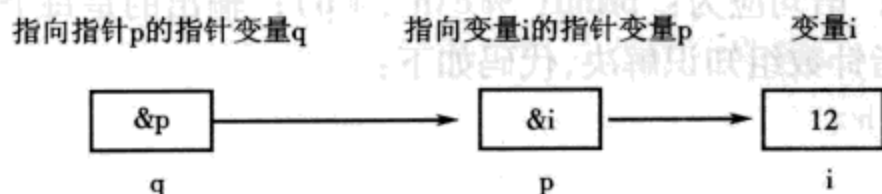


图 8.8 变量、指针变量与指向指针变量的指针变量的关系

【例 8.3】 使用指向指针的指针。

【参考代码】

```
#include <stdio.h>
void main()
{
    char name[5][20] = {"BASIC", "Great Wall", "JavaScript", "AUTOCAD", "Visual C++6.0"};
    char * p1, ** p;
    int i;
    for(i = 0; i < 5; i++)
    {
        p1 = name[i];
        p = &p1;
        printf("%s\n", *p);
    }
}
```

【程序说明】

- (1) “ $p1 = name[i];$ ”语句说明把第 i 个字符串的首地址赋给 $p1$;
- (2) “ $p = \&p1;$ ”语句把指针变量 $p1$ 的地址赋给了 p ;
- (3) “ $printf("%s\n", *p);$ ”输出 p 所指向的地址的地址单元的内容。
- (4) 本例可用如下代码片断实现:

```
char * p;
int i;
for(i = 0; i < 5; i++)
{
    p = name[i];
    printf("%s\n", p);
}
```

- (5) 不能使用如下代码片断实现:

```
char * p;
int i;
for(i = 0; i < 5; i++)
```

```

    p = name[i];
    printf("%s\n", *p);

```

因为 p 所指的是字符串 name[i] 的首地址, *p 所指单元内容应是 name[i][0] 的字符, “printf(“%s\n”, *p);”语句应为:“printf(“%c\n”, *p);”输出的是每个字符串的首字符了。

(6) 本例也可用指针数组知识解决,代码如下:

```

#include <stdio.h>

void main()
{
    char * name[] = {"BASIC", "Great Wall", "JavaScript", "AUTOCAD", "Visual C++6.0"};
    char ** p;
    int i;
    for(i = 0; i < 5; i++)
    {
        p = name + i;          //或 p = &name[i]; 不能为 p = name[i];
        printf("%s\n", *p);
    }
}

```

程序运行结果如图 8.9 所示。

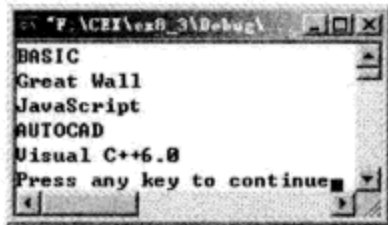


图 8.9 例 8.3 运行结果

8.2 指针变量与函数参数

8.2.1 指针作为函数参数

函数的参数不仅可以是整型、实型、字符型等数据,还可以是指针类型。它的作用是将一个变量的地址传送到另一个函数中。

【例 8.4】 用指针作为参数实现从键盘上输入两个数,并求其最大值。

【参考代码】

```

#include <stdio.h>

void max(int *p1, int *p2)
{
    int temp;
    if (*p1 > *p2)
    {
        temp = *p1;
        *p1 = *p2;
        //交换 p1, p2 指向的存储单元的内容
    }
}

```

```

    * p2 = temp;
}

void main()
{ int a,b;
  int * p, * q;
  printf("请输入两个数 a,b:");
  scanf("%d %d",&a,&b);
  p = &a; //p 指向 a
  q = &b; //q 指向 b
  max(p,q); //调用函数 max()
  printf("a,b 的最大值 = %d\n",b); //b 为 a,b 的最大值
}

```

程序运行结果如图 8.10 所示。

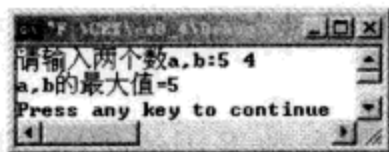


图 8.10 例 8.4 运行结果

【注意点】

(1) 主函数中的 `max(p,q)` 将实参 `p`、`q` 的地址分别传递给被调 `max()` 中的形参 `p1`、`p2`。
`printf("a,b 的最大值 = %d\n",b);` 实现了输出两个数的最大值,如图 8.11 所示。

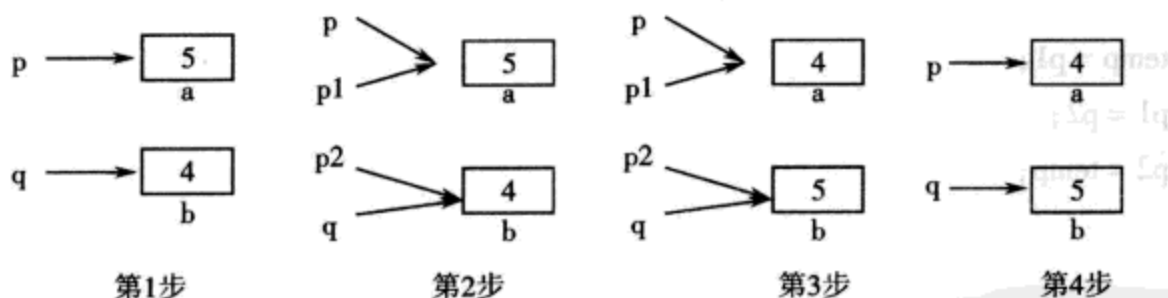


图 8.11 例 8.4 的执行过程

(2) 代码片断:

```

int temp;
if ( * p1 > * p2)
{ temp = * p1; * p1 = * p2; * p2 = temp; }

```

不能修改为:

```

int * temp;
if ( * p1 > * p2)
{ temp = p1; p1 = p2; p2 = temp; }

```

因为这个代码片断仅交换了两个指针变量的地址,没有实现 `* p1` 与 `* p2` 的交换,即 `a` 与 `b` 没有交换,请读者注意。其执行过程如图 8.12 所示。

(3) 本例可以用传值的方式解决,详见下小节。

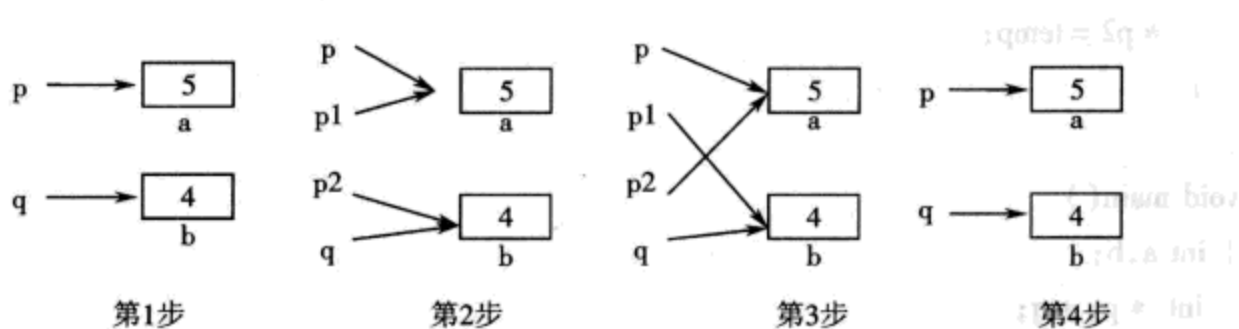


图 8.12 错误代码的执行过程

8.2.2 值传递与地址传递的区别与联系

在前面的章节中曾介绍过,当变量作为被调函数形参时,它是函数的局部变量,它在函数内的改变不会影响到主调函数对应实参变量的值。但如果用指针作为函数参数来传递,情况就不同了。上一节介绍了指针(地址)作为函数参数传递的方法,通过指针变量去改变对应变量的值会影响到主调函数对应变量的值。

【例 8.5】 用值作为参数实现从键盘上输入两个数,并求其最大值。

【解题思路】

如果按例 8.4 的格式修改一下,使参数采用值传递,即代码如下:

```
#include <stdio.h>
void max(int p1,int p2)
{
    int temp;
    if (p1 > p2)
    {
        temp = p1;
        p1 = p2;
        p2 = temp;
    }
}
void main()
{
    int a,b;
    int *p,*q;
    printf("请输入两个数 a,b:");
    scanf("%d%d",&a,&b);
    p = &a;
    q = &b;
    max(*p,*q);
    printf("a,b 的最大值 = %d\n",b);
}
```

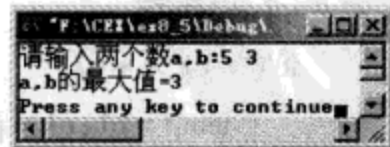


图 8.13 例 8.5 错误代码
执行结果

则其输出的结果如图 8.13 所示。

例 8.4 中函数参数采用地址传递,它传递的是变量的地址,因此在此函数中用此地址改变对应变量的值时,实际上改变的是主调

函数中变量的值。而例 8.5 的代码仅实现了函数内部形参的值的操作,根本没有对主调函数中实际参数的操作。其执行过程如图 8.14 所示。

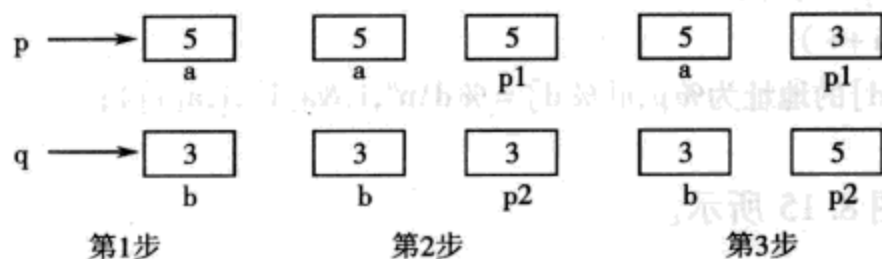


图 8.14 例 8.5 错误代码的执行过程

正确代码详见例 8.4 或修改如下:

```
#include <stdio.h>
int max(int p1, int p2)
{
    return(p1 > p2 ? p1 : p2);
}
void main()
```

```

{
    int a, b;
    int *p, *q;
    printf("请输入两个数 a,b:");
    scanf("%d %d", &a, &b);
    p = &a;
    q = &b;
    printf("a,b 的最大值 = %d\n", max(*p, *q));
}
```

8.3 指针与数组

8.3.1 指向数组的指针

一个变量有一个地址,一个数组包含若干元素,每个数组元素都在内存中占用存储单元,它们都有相应的地址。所谓数组的指针是指数组的起始地址,数组元素的指针是数组元素的地址。

一个数组是由连续的一段内存单元组成的。数组名就是这段连续内存单元的首地址。例如:

```
int a[4];
```

则 $a[0]$ 、 $a[1]$ 、 $a[2]$ 、 $a[3]$ 都是整型变量,既然是变量,它们在内存中的地址就可找到,也就是可以取出其指针地址,而且这些地址是连续的。

【例 8.6】 查看数组单元的地址。

【参考代码】

```
#include <stdio.h>
```



```

void main()
{
    int i, a[4] = {5, 6, 7, 8};
    for(i = 0; i < 4; i++)
        printf("a[%d]的地址为%p, a[%d] = %d\n", i, &a[i], i, a[i]);
}

```

程序运行结果如图 8.15 所示。

【注意点】

“printf(“a[%d]的地址为%p, a[%d] = %d\n”, i, &a[i], i, a[i]);”语句中的“%p”输出“&a[i]”的内存地址, 如果用“%X”代替“%p”, 则输出“&a[i]”的地址如图 8.16 所示, 比图 8.15 中的地址少两位十六进制数。

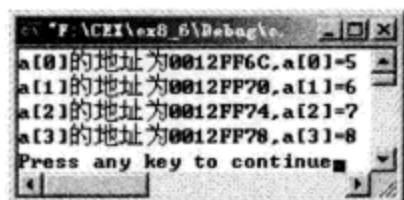


图 8.15 例 8.6 执行结果 1

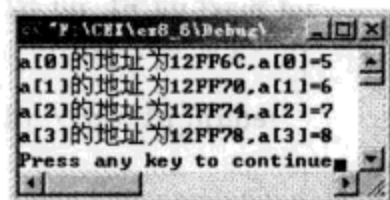


图 8.16 例 8.6 执行结果 2

8.3.2 通过数组指针访问数组

既然数组名就是数组的首地址, 也就是可以通过数组指针去访问数组中的任何一个单元, 方法是把数组的地址指针加上一个数组下标的整数, 得到的值就是数组的对应单元的地址。例如:

```
int a[4];
```

a 是数组的首地址, 即 &a[0], 而 a+1 是 a[1] 的地址, 即 &a[1], *(a+1) 是对应的单元变量, 即 a[1]。一般如果 i 是数组的整数下标, 则 a+i 是第 i 个单元(或元素)的地址, 即 a+i 就是 &a[i], 同时 *(a+i) 就是 a[i], 通过 *(a+i) 去存取数组元素单元的值与用 a[i] 去存取是完全相同的。

值得注意的是, a+i 不是简单地把 a 的地址值加上一个整数 i, 对于整数数组来说, 实际上 a+i 比 a 地址多了 sizeof(int) * i 的值。

【例 8.7】用数组地址表示法存取数组元素。

【参考代码】

```

#include <stdio.h>

void main()
{
    int i, a[4] = {5, 6, 7, 8};
    printf("用数组元素表示法存取数组元素:\n");
    for(i = 0; i < 4; i++)
        printf("a[%d] = %d\n", i, a[i]);
    printf("用数组地址表示法存取数组元素:\n");
    for(i = 0; i < 4; i++)

```

```
printf("a + %d = %p, a[%d] = %d\n", i, a + i, i, *(a + i));
```

程序运行结果如图 8.17 所示。

8.3.3 通过指针变量访问数组

定义一个指向数组元素的指针变量的方法,与以前介绍的指针变量相同。例如:

```
int a[4];    /* 定义 a 为包含 4 个整型数据的数组 */
int *p;      /* 定义 p 为指向整型变量的指针 */
```

应当注意,因为数组为 int 型,所以指针变量也应为指向 int 型的指针变量。下面是对指针变量赋值:

```
p = &a[0];
```

把 a[0] 元素的地址赋给指针变量 p。也就是说, p 指向 a 数组的第 0 号元素。

C 语言规定,数组名代表数组的首地址,也就是第 0 号元素的地址。因此,下面两个语句等价:

```
p = &a[0];
p = a;
```

在定义指针变量时可以赋给初值:

```
int *p = &a[0];
```

它等效于:

```
int *p;
p = &a[0];
```

当然定义时也可以写成:

```
int *p = a;
```

【例 8.8】用指针存取数组元素。

```
#include <stdio.h>
```

```
void main()
```

```
{
    int i, a[4] = {5, 6, 7, 10}, *p, *q;
```

```
    q = a + 1;
```

```
    p = a; printf("这是 a[0]: &a[0] = %p, a[0] = %d\n", &a[0], *p);
```

```
    ++p; printf("这是 a[1]: &a[1] = %p, a[1] = %d\n", &a[1], *p);
```

```
    --p; printf("这是 a[0]: &a[0] = %p, a[0] = %d\n", &a[0], *p);
```

```
    p = p + 3; printf("这是 a[3]: &a[3] = %p, a[3] = %d\n", &a[3], *p);
```

```
    *p = 8; printf("这是 a[3]: &a[3] = %p, a[3] = %d\n", &a[3], *p); // 修改过的 a[3] 的值
```

```
    p = p - 3; printf("这是 a[0]: &a[0] = %p, a[0] = %d\n", &a[0], *p);
```

```
    p = a + 3; printf("指针 p 与 q 的差: p - q = %d - %d = %d\n", p - a, q - a, p - q);
```

程序运行结果如图 8.18 所示。

本例指针操作数组元素过程如图 8.19 所示。

从本例可以看出,引入指针变量后,就可以用两种方法来访问数组元素了。

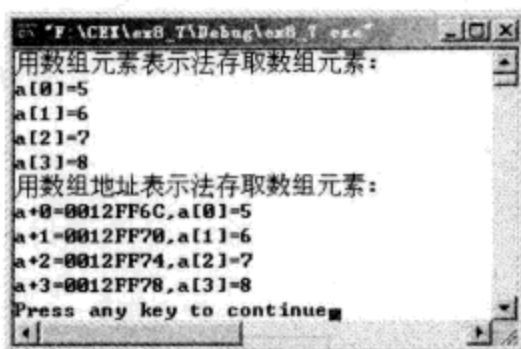


图 8.17 例 8.7 执行结果



图 8.18 例 8.8 执行结果

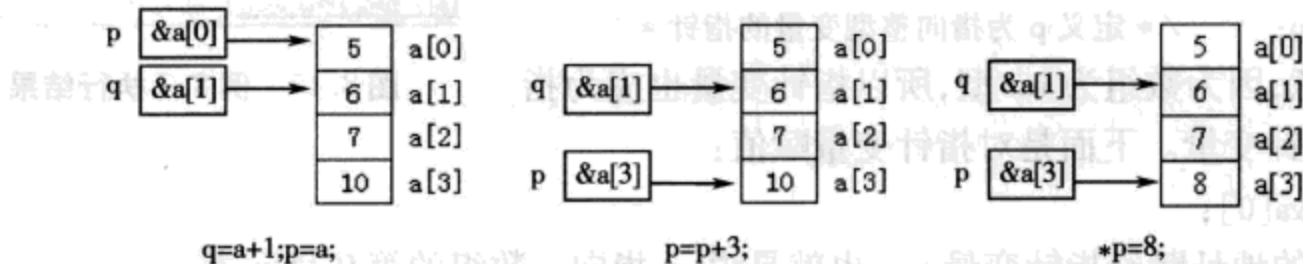


图 8.19 例 8.8 执行过程

如果 p 的初值为 $\&a[0]$, 则有以下注意事项。

- (1) $p+i$ 和 $a+i$ 都是 $a[i]$ 的地址, 或者说它们指向 a 数组的第 i 个元素。
- (2) $*(p+i)$ 或 $*(a+i)$ 就是 $p+i$ 或 $a+i$ 所指向的数组元素, 即 $a[i]$ 。例如, $*(p+3)$ 或 $*(a+3)$ 就是 $a[3]$ 。

(3) 指向数组的指针变量也可以带下标, 如 $p[i]$ 与 $*(p+i)$ 等价。

(4) 引用一个数组元素可以用下标法和指针法来完成。

① 下标法, 即用 $a[i]$ 形式访问数组元素。在前面介绍数组时都是采用这种方法。

② 指针法, 即采用 $*(a+i)$ 或 $*(p+i)$ 形式, 用间接访问的方法来访问数组元素, 其中 a 是数组名, p 是指向数组的指针变量, 其值 $p=a$ 。

注意, 指针变量可以实现本身的值的改变。如 $p++$ 是合法的; 而 $a++$ 是错误的。因为 a 是数组名, 它是数组的首地址, 是常量。同时, 要注意指针变量的当前值。

(1) 由于 $++$ 和 $*$ 同优先级, 结合方向自右而左, 所以, $*p++$ 等价于 $*(p++)$ 。

(2) $*(p++)$ 与 $*(++p)$ 作用不同。若 p 的初值为 a , 则 $*(p++)$ 等价 $a[0]$, $*(++p)$ 等价 $a[1]$ 。

(3) $(*p)++$ 表示 p 所指向的元素值加 1。

(4) 如果 p 当前指向 a 数组中的第 i 个元素, 则有: $*(p--)$ 相当于 $a[i--]$; $*(++p)$ 相当于 $a[++i]$; $*(--p)$ 相当于 $a[--i]$ 。

8.3.4 数组名作函数参数

在上一章中介绍过数组(名)作为函数的参数, 其一般形式为:

类型说明符 函数名(数组类型说明符 数组名[]);

本节要介绍的是数组名作为函数参数的另一形式, 其形式为:

类型说明符 函数名(数组类型说明符 *数组名);

例如:

```
int f(int a[ ]);
```

```
int f(int *a);
```

这两者是等价的。注意,在用第一种形式时不必在[]中写出形参数组的大小。在调用函数时,实参就是数组的名称,也就是数组指针。

【例 8.9】 输入 10 个数并存入一个数组中,利用数组作为函数参数的两种形式求 10 个数中的最大值。

【参考代码】

```
#include <stdio.h>
```

```
void fun1(int a[],int n)
```

```
{
```

```
int max,i;
```

```
max = a[0];
```

//将数组第 0 元素的值赋给 max

```
for(i=1;i<n;i++)
```

```
if(max<a[i])
```

//使用数组元素

```
max = a[i];
```

```
printf("在函数 fun1()中的数组 a[] 的最大值:%d\n\n",max);
```

```
}
```

```
void fun2(int *a,int n)
```

```
{
```

```
int max,i;
```

```
max = *a;
```

//将数组首地址单元的值赋给 max

```
for(i=1;i<n;i++)
```

```
if(max<*(a+i))
```

//使用数组指针

```
max = *(a+i);
```

```
printf("在函数 fun2()中的数组 a[] 的最大值:%d\n\n",max);
```

```
}
```

```
void main()
```

```
{
```

```
int a[10],i;
```

```
char ch;
```

```
printf("请输入 10 个数:\n");
```

```
for(i=0;i<10;i++)
```

```
scanf("%d",&a[i]);
```

```
printf("\n");
```

```
fflush(stdin);
```

```
do
```

```
{
```

```
printf("请选择:0:退出\n");
```

```
printf("请选择:1:fun1()中处理数组 a.\n");
```

```
printf("请选择:2:fun2()中处理数组 a.\n");
```

```
ch = getchar();
```

```
fflush(stdin);
```

```
if(ch=='0')
```

1000	1001	1000
1	2	3
1011	1010	1010
4	2	0
1001	1001	1001
7	8	0

图 8.31

```

        break;
    else if( ch == '1')
        fun1(a,10);
    else if( ch == '2')
        fun2(a,10);
    } while(1);
}

```

【程序说明】

(1) 本例中采用两个函数来处理一个数组中的最大值。

(2) 采用 do-while 循环来选择 0、1 和 2, 分别实现退出循环、fun1() 处理和 fun2() 处理数组中的最大值。

(3) 程序中使用函数 fflush(stdin) 来处理输入数据时的“回车”符, 否则循环提示会出现多次, 请读者自己上机体会。

程序的运行结果如图 8.20 所示。



图 8.20 例 8.9 执行结果

8.4 指针与二维数组

本小节以二维数组为例, 介绍多维数组的指针变量。

8.4.1 二维数组的地址

设有整型二维数组 a[3][3] 如下:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

它的定义为:

```
int a[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } }
```

设数组 a 的首地址为 1000, 各下标变量的首地址及其值如图 8.21 所示。

1000	1004	1008
1	2	3
1012	1016	1020
4	5	6
1024	1028	1032
7	8	9

图 8.21 一个二维数组实例

注意, Visual C++ 6.0 每个 int 型数据占 4 个字节。前面介绍过, C 语言允许把一个二维数组分解为多个一维数组来处理。因此数组 a 可分解为三个一维数组, 即 a[0]、a[1]、a[2]。每一个一维数组又含有三个元素。如图 8.22 所示。

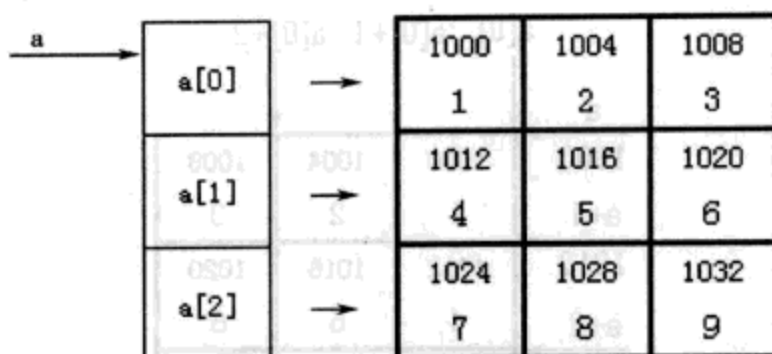


图 8.22 数组行地址

例如, $a[0]$ 数组含有 $a[0][0]$ 、 $a[0][1]$ 、 $a[0][2]$ 三个元素。

从二维数组的角度来看, a 是二维数组名, a 代表整个二维数组的首地址, 也是二维数组第 0 行的首地址, 等于 1000。 $a+1$ 代表第 1 行的首地址, 等于 1012, 依次类推……, 如图 8.23 所示。

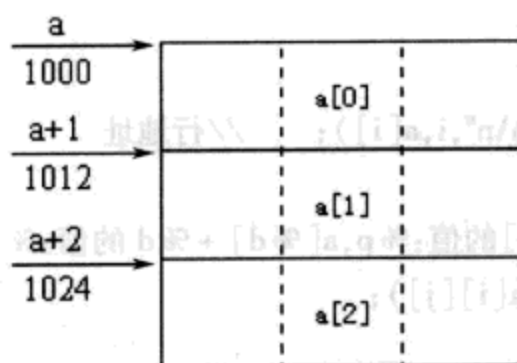


图 8.23 数组行指针

$a[0]$ 是第一个一维数组的数组名和首地址, 因此也为 1000。 $*(a+0)$ 或 $*a$ 是与 $a[0]$ 等效的, 它表示一维数组 $a[0]$ 的 0 号元素的首地址, 也为 1000。 $\&a[0][0]$ 是二维数组 a 的 0 行 0 列元素首地址, 同样是 1000。 因此, a 、 $a[0]$ 、 $*(a+0)$ 、 $*a$ 、 $\&a[0][0]$ 是相等的。

同理, $a+1$ 是二维数组第 1 行的首地址, 等于 1012。 $a[1]$ 是第二个一维数组的数组名和首地址, 因此也为 1012。 $\&a[1][0]$ 是二维数组 a 的 1 行 0 列元素地址, 也是 1012。 因此 $a+1$ 、 $a[1]$ 、 $*(a+1)$ 、 $\&a[1][0]$ 是等同的。

由此可得出: $a+i$ 、 $a[i]$ 、 $*(a+i)$ 、 $\&a[i][0]$ 是等同的。

此外, $\&a[i]$ 和 $a[i]$ 也是等同的。在二维数组中不能把 $\&a[i]$ 理解为元素 $a[i]$ 的地址, 因为不存在元素 $a[i]$ 。C 语言规定, 它是一种地址计算方法, 表示数组 a 第 i 行首地址。由此, 我们得出: $a[i]$ 、 $\&a[i]$ 、 $*(a+i)$ 和 $a+i$ 也都是等同的。

另外, $a[0]$ 也可以看成是 $a[0]+0$, 是一维数组 $a[0]$ 的 0 号元素的首地址, 而 $a[0]+1$ 则是 $a[0]$ 的 1 号元素地址, 由此可得出 $a[i]+j$ 则是一维数组 $a[i]$ 的 j 号元素地址, 它等于 $\&a[i][j]$, 如图 8.24 所示。

由 $a[i]$ 等价于 $*(a+i)$, 得 $a[i]+j$ 等价于 $*(a+i)+j$ 。由于 $*(a+i)+j$ 是二维数组 a 的 i 行 j 列元素的地址, 所以, 该元素的值等于 $*(*(a+i)+j)$ 。

【例 8.10】 利用二维数组指针操作数组。

```
#include <stdio.h>
```

```
void main()
```

		a[0]	a[0]+1	a[0]+2
a	1000	1000	1004	1008
a+1	1012	1	2	3
a+2	1024	4	5	6
		7	8	9

图 8.24 数组的指针

```

int a[3][3] = { 1,2,3,4,5,6,7,8,9 }, i, j;
printf("数组的地址及数组元素:\n");
for(i=0; i<3; i++)
{
    printf("a[ %d] 的值: %p\n", i, a[i]); //行地址
    for(j=0; j<3; j++)
        printf("&a[ %d][ %d] 的值: %p, a[ %d] + %d 的值: %p, a[ %d][ %d] = %d\n", i, j, &a[i][j], i, j, a[i] + j, i, j, a[i][j]);
}

printf("\n 用数组的指针形式表示数组元素的值:\n");
for(i=0; i<3; i++)
{
    printf("a[ %d] 的地址: %p\n", i, a+i);
    for(j=0; j<3; j++)
        printf("&a[ %d][ %d] 的值: %p, *(a+ %d) + %d 的值: %p, a[ %d][ %d] = %d\n", i, j, &a[i][j], i, j, *(a+i) + j, i, j, a[i][j]);
}

```

程序运行的结果如图 8.25 所示。

8.4.2 指向由 n 个元素组成的一维数组的指针变量

把二维数组 $a[3][3]$ 分解为一维数组之后, 设 p 为指向一维数组且有三个元素的指针变量。则可定义为:

```
int (*p)[3];
```

它表示 p 是一个指针变量, 它指向包含 3 个元素的一维数组。若指向第一个一维数组 $a[0]$, 其值等于 a 、 $a[0]$ 、 $\&a[0][0]$ 。而 $p+i$ 则指向一维数组 $a[i]$ 。从前面的分析可得出 $*(p+i)+j$ 是二维数组 i 行 j 列的元素的地址, 而 $*(*(p+i)+j)$ 则是 i 行 j 列元素的值。

指向由 n 元素组成的一维数组的指针变量一般形式为:

类型说明符 ($*$ 指针变量名)[长度];

其中“类型说明符”为所指数组的数据类型; “ $*$ ”表示其后的变量是指针类型; “长度”表示二

```

F:\VC6\bin\Debug\ex8_10.exe
数组的地址及数组元素:
a[0]的值:0012FF5C
a[0][0]的值:0012FF5C,a[0]+0的值:0012FF5C,a[0][0]=1
a[0][1]的值:0012FF60,a[0]+1的值:0012FF60,a[0][1]=2
a[0][2]的值:0012FF64,a[0]+2的值:0012FF64,a[0][2]=3
a[1]的值:0012FF68
a[1][0]的值:0012FF68,a[1]+0的值:0012FF68,a[1][0]=4
a[1][1]的值:0012FF6C,a[1]+1的值:0012FF6C,a[1][1]=5
a[1][2]的值:0012FF70,a[1]+2的值:0012FF70,a[1][2]=6
a[2]的值:0012FF74
a[2][0]的值:0012FF74,a[2]+0的值:0012FF74,a[2][0]=7
a[2][1]的值:0012FF78,a[2]+1的值:0012FF78,a[2][1]=8
a[2][2]的值:0012FF7C,a[2]+2的值:0012FF7C,a[2][2]=9

用数组的指针形式表示数组元素的值:
a[0]的地址:0012FF5C
a[0][0]的值:0012FF5C,*a[0]+0的值:0012FF5C,a[0][0]=*(a[0]+0)=1
a[0][1]的值:0012FF60,*a[0]+1的值:0012FF60,a[0][1]=*(a[0]+1)=2
a[0][2]的值:0012FF64,*a[0]+2的值:0012FF64,a[0][2]=*(a[0]+2)=3
a[1]的地址:0012FF68
a[1][0]的值:0012FF68,*a[1]+0的值:0012FF68,a[1][0]=*(a[1]+0)=4
a[1][1]的值:0012FF6C,*a[1]+1的值:0012FF6C,a[1][1]=*(a[1]+1)=5
a[1][2]的值:0012FF70,*a[1]+2的值:0012FF70,a[1][2]=*(a[1]+2)=6
a[2]的地址:0012FF74
a[2][0]的值:0012FF74,*a[2]+0的值:0012FF74,a[2][0]=*(a[2]+0)=7
a[2][1]的值:0012FF78,*a[2]+1的值:0012FF78,a[2][1]=*(a[2]+1)=8
a[2][2]的值:0012FF7C,*a[2]+2的值:0012FF7C,a[2][2]=*(a[2]+2)=9
Press any key to continue

```

图 8.25 例 8.10 运行结果

维数组分解为多个一维数组时,一维数组的长度,也就是二维数组的列数。应注意“(* 指针变量名)”两边的括号不可少,如缺少括号则表示是指针数组(本章后面介绍),意义就完全不同了。

【例 8.11】 用指向二维数组的指针变量输出数组。

【程序代码】

```

#include <stdio.h>
void main()
{
    int a[3][3] = { 1,2,3,4,5,6,7,8,9 }, i,j;
    int (*p)[3];    /* p 是一个指针变量,仅为一个变量,不能写成 int *p[3],它表示含 3
                     个元素的数组,而这 3 个元素是指针,且是指向整型数的指针,属于指针
                     数组。*/

    p = a;
    printf("用指向由 3 个元素组成的一维数组的指针变量输出数组元素:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
            printf("a[%d][%d] = %d ",i,j, *(*(p+i)+j));
        printf("\n");
    }
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
            printf("%d ", *(*(a+i)+j));
        printf("\n");
    }
}

```

程序的运行结果如图 8.26 所示。

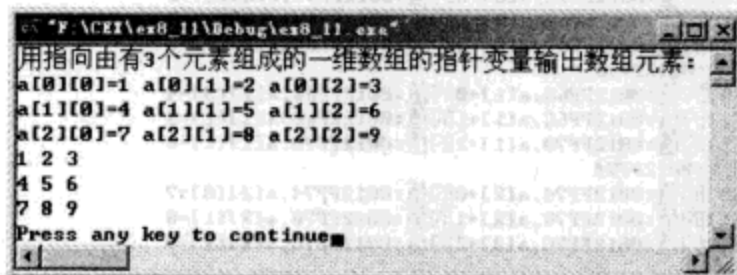


图 8.26 例 8.11 运行结果

8.4.3 指向二维数组元素的指针变量

上一节的指向由 n 个元素组成的一维数组的指针变量理解起来较难,这一节介绍的是指向二维数组元素的指针变量,理解起来比较容易。例如:

```
int a[3][3];
int *p;           //没有[]括号,也没有(*p)
```

通过下面例子即可体会二者的区别。

【例 8.12】 输出二维数组元素。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int a[3][3] = {1,2,3,4,5,6,7,8,9}, i=1;
```

```
int *p;
```

//是一个指针变量,与(*p)[n]不同

```
p = a[0];
```

```
printf("a 数组为:\n");
```

```
for(i=1; p < a[0] + 9; p++, i++)
```

```
printf("%4d", *p);
```

```
if(i%3 == 0)
```

```
printf("\n");
```

```
}
```

程序运行的结果如图 8.27 所示。

【注意点】

“ $p = a[0];$ ”语句如果写成: $p = a;$ 则会出现“warning C4047: ‘ $=$ ’: ‘ $\text{int} *$ ’ differs in levels of indirection from ‘ $\text{int} (*)[3]$ ’”警告信息,但运行结果相同。

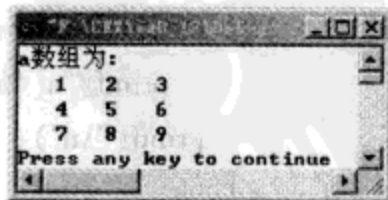


图 8.27 例 8.12 运行结果

8.5 字符串与指针

8.5.1 字符串的表示形式

前面学习了指针,按照所学知识,字符指针(char *)应该是指向字符变量。但实际应用中,人们常用字符指针指向字符数组的元素,以便通过这种指针使用字符数组的内容。最常见的情况是令字符指针指向字符串,该字符串可以是一个常量字符串,也可以是一个存储着字符串的字符数组。

在C语言中,可以用两种方法访问一个字符串。

1. 用字符指针指向一个字符串

例如:

```
char *p = "This is a C Language";
```

该语句首先定义了字符指针p,然后建立一个字符串常量"This is a C Language",它以字符数组形式存储,最后有一个空字符'\0';给p赋初值,使它指向刚建立的字符串常量,如图8.28所示。

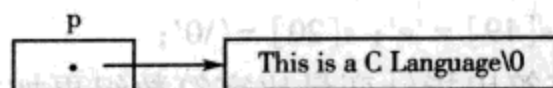


图 8.28 字符串与字符指针

【例 8.13】 输出上面的字符串。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    char *p = "This is a C Language";
```

```
    printf("%s\n", p);
```

```
}
```

说明:与前面介绍的数组属性一样,p是字符指针变量,字符串的首地址赋给p,如图8.29所示。

2. 用字符数组指向一个字符串

例如:

```
char p[] = "This is a C Language";
```

【例 8.14】 输出上面的字符串。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    char p[] = "This is a C Language";
```

```
    printf("%s\n", p);
```

```
}
```

3. 字符串指针变量与字符数组的区别

用字符数组和字符串指针变量都可以实现字符串的存储和运算。但是两者是有区别的,

在使用时应注意以下几个方面的问题。

(1) 字符串指针变量是一个变量,用于存放字符串的首地址。而字符串本身是存放在以该首地址为起点的一段连续的内存空间中,并以“\0”作字符串的结束标志。字符数组是由若干个数组元素组成的,它可以用来存放整个字符串,不一定有结束标志“\0”。

(2) 对字符串指针的操作方式。

例如:

```
char *s = "This is a C Language";
```

可以写成:

```
char *s;
```

```
s = "This is a C Language";
```

而对数组:

```
char s[] = "This is a C Language";
```

不能写成:

```
char s[21];
```

```
s = "This is a C Language";
```

而只能对字符数组的各个数组元素逐个赋值。即:

```
s[0] = 'T'; s[1] = 'h'; .....; s[19] = 'e'; s[20] = '\0';
```

从上面两点可以看出使用字符串指针变量比字符数组更加方便。

T	p[0]
h	p[1]
i	p[2]
s	p[3]
	p[4]
i	p[5]
s	p[6]
	p[7]
a	p[8]
	p[9]
C	p[10]
	p[11]
L	p[12]
a	p[13]
n	p[14]
g	p[15]
u	p[16]
a	p[17]
g	p[18]
e	p[19]
\0	p[20]

图 8.29 字符串表示

8.5.2 字符数组与字符串指针作函数参数

字符数组与字符串指针可以作为函数的参数传递。

1. 用字符数组作函数参数

【例 8.15】 本例把字符数组作为函数参数使用。要求把一个字符串的内容复制到另一个字符串中,并且不能使用 strcpy() 函数。函数 str_cpy 的形参为两个字符数组。s1 是源字符数组, s2 是目标字符数组。

【参考代码】

```
#include <stdio.h>
void str_cpy (char p1[], char p2[])
{
    int i;
    for(i=0; i <= strlen(p1) - 1; i++)
    {
        p2[i] = p1[i];
    }
    p2[i] = '\0';
}
void main()
{
    char s1[] = "This is a C Language", s[30];
    str_cpy(s1, s);
```

```
printf("源 串 = %s\n 目的串 = %s\n", s1, s);
```

程序运行结果如图 8.30 所示。

【注意点】

程序片断:

```
for(i=0; i <= strlen(p1) - 1; i++)
```

```
    p2[i] = p1[i];
```

```
    p2[i] = '\0';
```

可以书写为:

```
for(i=0; i <= strlen(p1); i++) { p2[i] = p1[i]; }
```

这样更简洁。

2. 用字符串指针作函数参数

【例 8.16】 本例是把字符串指针作为函数参数使用。要求把一个字符串的内容复制到另一个字符串中,并且不能使用 `strcpy()` 函数。函数 `strcpy()` 的形参为两个字符指针变量。`p1` 指向源字符串, `p2` 指向目标字符串。

【解题思路】

注意表达式: `(*p2 = *p1) != '\0'` 的用法。

【参考代码】

```
#include <stdio.h>
```

```
void strcpy (char * p1, char * p2)
```

```
{
    while( *p1 != '\0')
```

```
        *p2 = *p1;
```

```
        p2 ++;
```

```
        p1 ++;
```

```
    *p2 = '\0';
}
```

```
void main()
```

```
{
    char * p1 = "This is a C Language", s[30], * p2;
```

```
    p2 = s;
```

```
    strcpy(p1, p2);
```

```
    printf("源 串 = %s\n 目的串 = %s\n", p1, p2);
}
```

程序运行结果也如图 8.30 所示。

【注意点】

(1) 在字符串的复制中可用如下语句来复制字符串,这样较简洁。

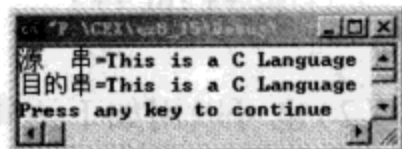


图 8.30 例 8.15 执行结果

```
while(( * p2 = * p1) != '\0')
```

```
{
```

```
    p2 ++ ; p1 ++ ;
```

```
}
```

(2) 也可以用如下语句来复制字符串, 这样比上面更简洁。

```
while(( * p2 ++ = * p1 ++ ) != '\0')
```

8.6 指针数组

由若干个指针变量组成的数组称为指针数组。指针数组也是一种数组, 所有有关数组的概念都适用于它。但是指针数组与普通数组又有区别, 它的数组元素是指针类型的, 只能用来存放地址值。也就是说, 指针数组是一组有序的指针的集合。指针数组的所有元素都是具有相同存储类型和指向相同数据类型的指针变量。

指针数组定义的一般形式为:

类型说明符 * 数组名[长度];

其中“类型说明符”为指针值所指向的变量的类型。例如:

```
int * p[4];
```

该语句表示 p 是一个指针数组, 它有 4 个元素, 每个元素的值都是一个指针, 且指向整型量(常量、变量或表达式)。

通常可用一个指针数组来指向一个二维数组, 指针数组中的每个元素被赋予二维数组每一行的首地址, 因此, 也可以理解为指向一个一维数组。

应该注意, 指针数组和指向由 n 个元素组成的一维数组的指针变量的区别。这两者虽然都可用来指向二维数组, 但是其表示方法和意义是不同的。

指向由 n 个元素组成的一维数组的指针变量是单个的变量, 其一般形式中“(* 指针变量名)”两边的括号不可少。而指针数组表示的是多个指针(一组有序指针), 其一般形式中“* 指针数组名”两边不能有括号。例如:

```
int (* p)[4];
```

表示一个指向由 n 个元素组成的一维数组的指针变量, 该一维数组的列数为 4, p 是分解为一维数组的且长度为 4 的行指针。而

```
int * p[4];
```

表示 p 是一个指针数组, 有 4 个下标元素, p[0]、p[1]、p[2]、p[3] 均为指针变量。

【例 8.17】 使用指针数组操作二维数组。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
    int a[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

```
    int * p[4], i, j;
```

```
    int * q = a[0];          //q 指针赋初值 a[0]
```

```
    for(i=0; i<3; i++)        //给指针数组 * p[4] 赋值
```

```

p[i] = a[i];
printf("用指针数组表示数组元素的值:\n");
for(i=0;i<3;i++)
{
    printf("a[%d]的地址:%p,a[%d][%d] = *p[%d] = %d\n",i,p[i],i,i,*p[i]);
    printf("a[%d]的地址 = *(p+%d):%p,a[%d][%d] = *(*(p+%d)) = %d\n",i,*(p+i),i,i,i,*(*(p+i)));
    //p[i]表示第i行首地址,则*p[i]为第i行首元素的值
    for(j=0;j<4;j++)
        printf("&a[%d][%d] = p[%d] + %d = %p,p[%d] + %d 的值:%p,a[%d][%d] = *(p[%d] + %d) = %d\n",i,j,i,j,p[i]+j,i,j,p[i]+j,i,j,i,j,*(p[i]+j));
    printf("\n 用 *(p+i) + j 表示 &a[i][j], *(*(p+i) + j) 表示 a[i][j]:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
            printf("&a[%d][%d] = *(p+%d) + %d 的值:%p,a[%d][%d] = *(*(p+%d) + %d) = %d\n",i,j,i,j,*(p+i)+j,i,j,i,j,*(*(p+i)+j));
        printf("\n");
    }
    printf("*p 表示数组首地址:%p\n",*p);
}

```

程序运行结果如图 8.31 所示。

【注意点】

(1) p 是一个指针数组, p 代表指针数组的首地址, $*p$ 代表 $p[0]$ 或 $a[0]$ 的地址, 不能理解为 $a[0][0]$ 的值; $*(*p)$ 才代表 $a[0][0]$ 的值。

(2) $p[i]$ 代表数组第 i 行的首地址, $*p[i]$ 则代表数组第 i 行首元素的值, 即 $a[i][0]$ 的值。

(3) $*(p+i)$ 也代表数组第 i 行首地址, 等价于 $p[i]$, 而 $*(*(p+i))$ 则代表 $a[i][0]$ 的值。

(4) $p[i] + j$ 代表数组第 i 行 j 列的地址, $*(p[i] + j)$ 则代表数组第 i 行 j 列的元素 $a[i][j]$ 的值。

(5) $*(p+i) + j$ 也代表数组第 i 行 j 列的地址, $*(*(p+i) + j)$ 也代表数组第 i 行 j 列的 $a[i][j]$ 的值。

请读者仔细体会上述各点。

从上例可以理解指针数组知识。但实际上使用指针数组最重要的方面是, 要体现指针数组在处理字符串方面的优势, 下面一个例子就很有代表性。

指针数组也常用来指向一组字符串, 这时指针数组的每个元素被赋予一个字符串的首地址。指向一组字符串的指针数组的初始化更为简单。例如例 8.3 现可以采用指针数组来表示一组字符串。其初始化赋值为:

```

"F:\CEI\ex8_17\Debug\ex8_17.exe"
用指针数组表示数组元素的值:
a[0]的地址:0012FF50,a[0][0]=*p[0]=1
a[0]的地址=*(p+0):0012FF50,a[0][0]=*(p+0)=1
&a[0][0]=p[0]+0=0012FF50,p[0]的值:0012FF50,a[0][0]=*(p+0)=1
&a[0][1]=p[0]+1=0012FF54,p[0]的值:0012FF54,a[0][1]=*(p+0)+1=2
&a[0][2]=p[0]+2=0012FF58,p[0]的值:0012FF58,a[0][2]=*(p+0)+2=3
&a[0][3]=p[0]+3=0012FF5C,p[0]的值:0012FF5C,a[0][3]=*(p+0)+3=4
a[1]的地址:0012FF60,a[1][1]=*p[1]=5
a[1]的地址=*(p+1):0012FF60,a[1][1]=*(p+1)=5
&a[1][0]=p[1]+0=0012FF60,p[1]的值:0012FF60,a[1][0]=*(p+1)=5
&a[1][1]=p[1]+1=0012FF64,p[1]的值:0012FF64,a[1][1]=*(p+1)+1=6
&a[1][2]=p[1]+2=0012FF68,p[1]的值:0012FF68,a[1][2]=*(p+1)+2=7
&a[1][3]=p[1]+3=0012FF6C,p[1]的值:0012FF6C,a[1][3]=*(p+1)+3=8
a[2]的地址:0012FF70,a[2][2]=*p[2]=9
a[2]的地址=*(p+2):0012FF70,a[2][2]=*(p+2)=9
&a[2][0]=p[2]+0=0012FF70,p[2]的值:0012FF70,a[2][0]=*(p+2)=9
&a[2][1]=p[2]+1=0012FF74,p[2]的值:0012FF74,a[2][1]=*(p+2)+1=10
&a[2][2]=p[2]+2=0012FF78,p[2]的值:0012FF78,a[2][2]=*(p+2)+2=11
&a[2][3]=p[2]+3=0012FF7C,p[2]的值:0012FF7C,a[2][3]=*(p+2)+3=12

用*(p+i)+j表示&a[i][j],*(p+i)+j表示a[i][j]:
&a[0][0]=*(p+0)+0的值:0012FF50,a[0][0]=*(p+0)+0=1
&a[0][1]=*(p+0)+1的值:0012FF54,a[0][1]=*(p+0)+1=2
&a[0][2]=*(p+0)+2的值:0012FF58,a[0][2]=*(p+0)+2=3
&a[0][3]=*(p+0)+3的值:0012FF5C,a[0][3]=*(p+0)+3=4

&a[1][0]=*(p+1)+0的值:0012FF60,a[1][0]=*(p+1)+0=5
&a[1][1]=*(p+1)+1的值:0012FF64,a[1][1]=*(p+1)+1=6
&a[1][2]=*(p+1)+2的值:0012FF68,a[1][2]=*(p+1)+2=7
&a[1][3]=*(p+1)+3的值:0012FF6C,a[1][3]=*(p+1)+3=8

&a[2][0]=*(p+2)+0的值:0012FF70,a[2][0]=*(p+2)+0=9
&a[2][1]=*(p+2)+1的值:0012FF74,a[2][1]=*(p+2)+1=10
&a[2][2]=*(p+2)+2的值:0012FF78,a[2][2]=*(p+2)+2=11
&a[2][3]=*(p+2)+3的值:0012FF7C,a[2][3]=*(p+2)+3=12

*p表示数组首地址:0012FF50

```

图 8.31 例 8.17 执行结果

```

char * name[] = { "BASIC",
                  "Great Wall",
                  "JavaScript",
                  "AUTOCAD",
                  "Visual C++ 6.0"};

```

完成这个初始化赋值之后, name[0] 即指向字符串 "BASIC", name[1] 指向 "Great Wall", ……。

【例 8.18】 使用指针数组操作字符串数组。输入 5 个国名并按字母顺序排列后输出。

【参考代码】

```

#include <stdio.h>
#include <string.h>
void sort(char * name[], int n)
{
    char * p;
    int i, j, k;
    for(i = 0; i < n - 1; i++) {
        k = i;
        for(j = i + 1; j < n; j++)
            if(strcmp(name[k], name[j]) > 0)
                k = j;
        if(k != i)
            //实现地址交换
    }
}

```



```

    p = name[i]; //把 name[i] 的地址赋给 p
    name[i] = name[k]; //把 p 的地址赋给 name[k]
    name[k] = p;
}

void main()
{
    static char * countryname[] = {"Japan", "China", "England", "France", "Costa Rica"};
    int n = 5, i;
    sort(countryname, n);
    for (i = 0; i < n; i++)
        printf("%s\n", countryname[i]);
}

```

程序运行结果如图 8.32 所示。

【程序说明】

(1) 在以前的例子中采用了普通的排序方法, 逐个比较之后交换字符串的位置。交换字符串的物理位置是通过字符串复制函数完成的。反复的交换将使程序执行的速度很慢, 同时由于各字符串(国名)的长度不同, 又增加了存储管理的负担。用指针数组能很好地解决这些问题。

(2) 在 sort() 函数中, 对两个字符串比较, 采用了 strcmp() 函数。strcmp() 函数允许参与比较的字符串以指针方式出现。name[k] 和 name[j] 均为指针, 因此是合法的。字符串比较后需要交换时, 只交换指针数组元素的值, 而不交换具体的字符串, 这样将大大减少运行时间, 提高了运行效率。

【注意点】

(1) 以下程序片断:

```

p = name[i];
name[i] = name[k];
name[k] = p;

```

因为 name[i] 与 name[k] 都是指针数组元素, 存放的是字符串的首地址, 他们交换的是地址, 而不是交换字符串。

(2) 指针数组定义时采用 static, 实际上不用也可以实现。

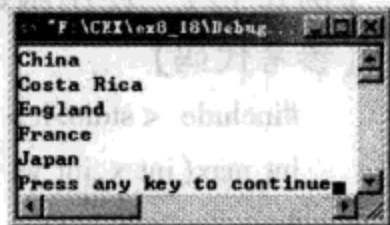


图 8.32 例 8.18 执行结果

8.7 指针与函数

8.7.1 函数指针

函数的指针就是函数的入口地址(函数的首地址)。C 语言规定函数的首地址就是函数名, 所以函数名就是函数的指针。

函数的指针可以用一个指向函数的指针变量来表达,这个指针变量用于存放函数的入口地址(函数指针或函数首地址),称为函数的指针变量。函数可以通过函数名调用,也可以通过函数指针调用。

通过函数指针实现函数调用的步骤如下。

(1) 定义指向函数的指针变量。其定义一般形式为:

类型说明符 (* 函数指针变量名)();

例如:

```
int (*p)();
```

注意,两组括号()都不能少,int 表示被指向函数的类型,即被指向函数的返回值的类型。

(2) 为指向函数的指针变量赋值,指向某函数。

函数指针变量名 = 函数名;

(3) 利用指向函数的指针变量调用函数:

(* 函数指针变量名)(实参列表);

【例 8.19】 使用函数名调用函数,实现输入两个整数,求其最大值。

【参考代码】

```
#include <stdio.h>
```

```
int max(int x,int y)
```

```
{
```

```
    return(x > y? x:y);
```

```
}
```

```
void main()
```

```
{
```

```
    int a,b;
```

```
    printf("输入两个整数 a,b:");
```

```
    scanf("%d%d",&a,&b);
```

```
    printf("a,b 的最大值为:%d\n",max(a,b));
```

```
}
```

程序运行结果如图 8.33 所示。

【例 8.20】 使用函数指针变量调用函数,实现输入两个整数,求其最大值。

【参考代码】

```
#include <stdio.h>
```

```
int max(int x,int y)
```

```
{
```

```
    return(x > y? x:y);
```

```
}
```

```
void main()
```

```
{
```

```
    int a,b,c;
```

```
    int (*f)() = max;
```

```
    printf("输入两个整数 a,b:");
```

```
    //输出提示行
```

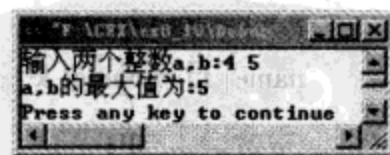


图 8.33 例 8.19 和例 8.20 执行结果

```
scanf("%d%d",&a,&b);
c = f(a,b);
printf("a,b 的最大值为:%d\n",c);
```

程序运行结果如图 8.33 所示。

【注意点】

(1) 使用函数指针变量的调用格式为:

$c = f(a,b)$; 或 $c = (*f)(a,b)$;

或 $c = (f)(a,b)$;

(2) 程序片断: “ $c = f(a,b)$; $\text{printf}("a,b \text{ 的最大值为: } \%d\backslash n", c)$ ”; 可以写成:

“ $\text{printf}("a,b \text{ 的最大值为: } \%d\backslash n", f(a,b))$ ”;

(3) 函数指针变量的赋值, 只需给出函数名, 不必给出参数。

(4) 只需理解这方面知识, 而不需要刻意使用函数指针变量。

8.7.2 返回值是指针的函数

函数可以返回整型、实型、字符型等类型的数据, 还可以返回地址值, 即返回指针值。

返回指针值的函数的定义一般形式为:

类型说明符 * 函数名(形参列表)

例如:

```
int * f(int x, int y)
```

表示函数 f 是返回整型指针的函数, 返回的指针值指向一整型数据。该函数还包含两个整型形参 x, y 。

【例 8.21】 编写两个字符串连接的程序, 相当于 $\text{strcat}()$ 函数的功能。

【参考代码】

```
#include <stdio.h>
char * strlink(char * s1, char * s2)
```

```
{
    char * temp;
    temp = s1;
    while(*s1)
        s1++;
    while(*s2 != '\0')
```

```
    *s1++ = *s2++;
```

```
    *s1 = '\0';
```

```
    return temp;
```

```
void main()
```

```
{
    char s3[80], s4[80], *s, *s1, *s2; s1 = s3; s2 = s4;
```

```

printf("输入两串字符 s1,s2:\n");
fflush(stdin);
gets(s1);
fflush(stdin);
gets(s2);
fflush(stdin);
s = strlink(s1,s2);
printf("连接后 s1 为:%s\n",s);

```

程序运行结果如图 8.34 所示。

【注意点】

如果用下列语句

```
char *s1, *s2, *s;
```

代替

```
char s3[80],s4[80], *s, *s1, *s2; s1 = s3; s2 = s4;
```

就不正常,但在 Turbo C 2.0 中正常。Visual C++ 6.0 编译时出现如下警告:

warning C4700: local variable 's1' used without having been initialized

warning C4700: local variable 's2' used without having been initialized

表示 s1,s2 没有分配内存空间,运行时也不正常。

如果在语句“char *s1, *s2, *s;”定义之后,使用如下语句:

```
s1 = (char *) malloc(80);
```

```
s2 = (char *) malloc(80);
```

即各分配一个 80 字节的内存空间,则程序编译不会出现警告语句。但要在程序最前面加上“#include <stdlib.h>”。

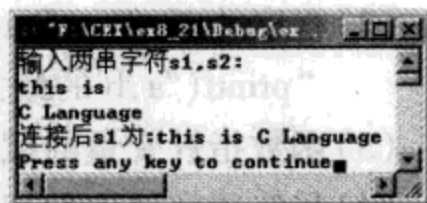


图 8.34 例 8.21 执行结果

8.8 main() 函数的参数

前面程序中,main()函数后面圆括号内都是空的,没有参数。其实,main()函数也可以有参数。例如:

```
main(int argc, char * argv[])
```

下面对 main()函数中参数进行如下说明。

(1) main()中参数只能有两个,且第一个参数必须是 int 型,第二个参数是 char 型的指针数组或 char 型的指向指针的指针变量。如 char * argv[] 或 char ** argv。

(2) 对于整型变量,建议使用 argc 作为变量名,另一个变量建议使用 argv,这两个参数名称是专门为 main()保留的,但也允许用户自己定义其他变量名。

(3) main()函数中参数的值是按如下方法得到的。

① 一个 C 语言程序经编译连接后生成 .exe 文件,这个 .exe 文件便可以在 DOS 环境下直接运行,这样 main()函数中参数的值便可由执行这个文件的命令提供。

② 在 DOS 环境下执行某个文件时,如 A1.exe,文件名后面可以跟由空格隔开的若干个字

字符串,如输入“A1.exe abcde 1234”并回车,则这个命令行中共有3个字符串(包括可执行文件名),则数值“3”传递给 main() 中的整型变量,如 argc,参数 char * argv[] 等价于 char * argv[3];3个字符串首地址分别作为指针数组 argv 中3个元素的值。

【例 8.22】 假定以下程序经编译和连接后生成可执行文件 PROG.EXE,如果在这个可执行文件所在目录的 DOS 提示符下键入:

PROG ABCDEFGH IJKL <回车>,观察输出结果。

//PROG.EXE

#include <stdio.h>

void main(int argc, char * argv[])

```
{
    while( --argc > 0)
        printf("%s", argv[argc]);
    printf("\n");
}
```

程序运行结果为:IJKLABCDEFGH

8.9 案例应用举例

【例 8.23】 利用指针变量与指针数组编写“图书管理系统”的“增加图书”子模块,实现图书信息的录入与输出。

【解题思路】

图书信息详见数组一章案例应用举例,要求录入 N 本书信息,书号、剩余本数为整型数组,对应指针采用指向整型数的指针变量 * p1, * p6;书名、作者与出版社为字符串数组,采用指向字符串数组的指针数组 * p2[N], * p3[N], * p4[N];书价为实型数组,对应指针采用指向实型的指针变量 * p5。

【参考代码】

#include <stdio.h>

#define N 2

void main()

```
{
    int booknum[N], * p1, * p6;
    char bookname[N][20], * p2[N];
    char bookauthor[N][20], * p3[N];
    char press[N][50], * p4[N];
    float price[N], * p5;
    int count[N];
    int i;
    p1 = booknum;
    p5 = price;
    p6 = count;
    for(i=0; i<N; i++)
```



```

p2[i] = bookname[i];
p3[i] = bookauthor[i];
p4[i] = press[i];
}
for(i=0;i<N;i++)
{
    printf("请输入第%d本书的信息:\n",i+1);
    printf("书号:");
    scanf("%d",p1+i);
    fflush(stdin);
    printf("书名:");
    gets(p2[i]);
    fflush(stdin);
    printf("作者:");
    gets(p3[i]);
    fflush(stdin);
    printf("出版社:");
    gets(p4[i]);
    fflush(stdin);
    printf("书价:");
    scanf("%f",p5+i);
    fflush(stdin);
    printf("剩余本数:");
    scanf("%d",p6+i);
    fflush(stdin);
}
printf("输出录入的图书信息:\n");
printf("书号    书名    作者    出版社    书价    剩余本数\n");
for(i=0;i<N;i++)
{
    printf("%4d",*(p1+i));
    printf("%8s",p2[i]);
    printf("%10s",p3[i]);
    printf("%9s",p4[i]);
    printf("%7.1f",*(p5+i));
    printf("%6d\n",*(p6+i));
}
}

```

本章小结

本章主要讲解了指针的基本概念和操作指针基本方法,它是本书的重点和难点。其中,指

针的含义与使用、函数之间的指针传递是本章的重点;而有关指针与数组、指针与函数、指针与字符串的关系及其应用,则是本章的难点。

习题八

一、选择题

1. 若 `char s[10]`, `*p = s`; , 则下列语句错误的是()。

- A) `p = s + 5` B) `s = [p + s]` C) `s[2] = p[4]` D) `*p = s[0]`

2. 若有定义: `int x`, `*p`; 则以下正确的赋值表达式是()。

- A) `p = &x` B) `p = x` C) `*p = &x` D) `*p = *x`

3. 已知定义“`char **s`;”, 下列语句正确的是()。

- A) `s = "computer"` B) `*s = "computer"` C) `**s = "computer"` D) `*s = 'A'`

4. 以下程序的输出结果是()。

```
#include <stdio.h>
void main()
{ printf("%d\n", NULL); }
```

- A) 因变量无定义输出不定值 B) 0
C) -1 D) 1

5. 若定义 `int a[3][4]`; 则下列选项不能表示 `a[1][1]` 的是()。

- A) `*(a[1] + 1)` B) `*(&a[1][1])` C) `((*(a + 1))[1])` D) `*(a + 1)`

6. 以下程序的输出结果是()。

```
#include <stdio.h>
void sub(int x, int y, int *z)
{ *z = y - x; }
void main()
{ int a, b, c;
  sub(10, 5, &a);
  sub(7, a, &b);
  sub(a, b, &c);
  printf("%d, %d, %d\n", a, b, c);
}
```

- A) 5, 2, 3 B) -5, -12, -7 C) -5, -12, -7 D) 5, -2, -7

7. 语句 `int (*p)()` 的含义是()。

- A) `p` 是一个指针型函数, 返回值为指针
B) `p` 是指针变量, 它指向一个整型数据的指针
C) `p` 是一个指向函数的指针, 该函数的返回值为整型
D) 以上答案都不对

8. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```

void main()
{
    int k=2, m=4, n=6;
    int *pk=&k, *pm=&m, *p;
    *(p=&n) = *pk * (*pm);
    printf("%d\n", n);
}

```

A)4

B)6

C)8

D)10

9. 下列程序执行后的输出结果是()。

```

#include <stdio.h>
void func(int *a, int b[])
{
    b[0] = *a + 6;
}
void main()
{
    int a, b[5];
    a = 0;
    b[0] = 3;
    func(&a, b);
    printf("%d\n", b[0]);
}

```

A)6

B)7

C)8

D)9

10. 有如下程序

```

#include <stdio.h>
void main()
{
    char ch[2][5] = {"6937", "8254"}, *p[2];
    int i, j, s = 0;
    for(i = 0; i < 2; i++)
        p[i] = ch[i];
    for(i = 0; i < 2; i++)
        for(j = 0; p[i][j] > '\0'; j += 2)
            s = 10 * s + p[i][j] - '0';
    printf("%d\n", s);
}

```

该程序的输出结果是()。

A)69825

B)63825

C)6385

D)693825

11. 已知指针 P 的指向如图 8.35 所示, 则表达式 *P++ 的值是()。

A)20

B)30

C)21

D)31

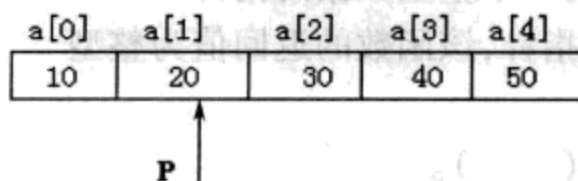


图 8.35 指针 P 指向图

12. 已知指针 P 的指向如图 8.35 所示, 则表达式 * ++P 的值是()。

- A) 20 B) 30 C) 21 D) 31

13. 有以下程序:

```
#include <stdio.h>
void ss(char *s, char t)
{ while(*s)
  { if(*s == t) *s = t - 'a' + 'A';
    s++; }
}
void main()
{ char str1[100] = "abcdddfefdbd", c = 'd';
  ss(str1, c);
  printf("%s\n", str1);
}
```

程序运行后的输出结果是()。

- A) ABCDDEFEDBD B) abcDDfefDbD C) abcAAfefAbA D) Abcddfefdbd

14. 以下程序的输出结果是()。

```
#include <stdio.h>
void prtv(int *x)
{ printf("%d\n", ++*x);
}
void main()
{ int a = 25;
  prtv(&a);
}
```

- A) 23 B) 24 C) 25 D) 26

15. 下面程序的输出结果是()。

```
#include <stdio.h>
void main()
{ int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, *p = a;
  printf("%d\n", *(p+2));
}
```

- A) 3 B) 4 C) 1 D) 2

16. 以下程序的输出结果是()。

```
#include <stdio.h>
void main()
{ int **k, b = 100, *j;
```

```

j = &b;
k = &j;
printf("%d\n", **k);

```

A) 运行出错 B) 100

C) a 的地址 D) b 的地址

17. 下面程序段的运行结果是()。

```

char str[] = "ABC", *p = str;
printf("%d\n", *(p+3));

```

A) 67 B) 0

C) 字符'C'的地址 D) 'C'

18. 以下程序的输出结果是()。

```

#include <stdio.h>
void fun(float *a, float *b)

```

```

{
    float w;
    *a = *a + *a;
    w = *a;
    *a = *b;
    *b = w;
}

```

```

void main()

```

```

{
    float x = 2.0, y = 3.0;
    float *px = &x, *py = &y;
    fun(px, py);
    printf("%2.0f, %2.0f\n", x, y);
}

```

A) 4, 3

B) 2, 3

C) 3, 4

D) 3, 2

19. 以下程序的输出结果是()。

```

#include <stdio.h>
void sub(float x, float *y, float *z)
{
    *y = *y - (float)1.0;
    *z = *z + x;
}

```

```

void main()

```

```

{
    float a = 2.5, b = 9.0, *pa, *pb;
    pa = &a; pb = &b;
    sub(b - a, pa, pb);
}

```



```
printf("%f\n", a);
```

- A) 9.000000 B) 1.500000 C) 8.000000 D) 10.500000

20. 下面选项正确的是()。

A) char *a = "china"; 等价于 char *a; *a = "china";

B) char str[5] = {"china"}; 等价于 char str[] = {"china"};

C) char *s = "china"; 等价于 char *s; s = "china";

D) char c[4] = "abc", d[4] = "abc"; 等价于 char c[4] = d[4] = "abc";

21. 若已定义:

```
int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, *p = a, i;
```

其中 $0 \leq i \leq 9$, 则对 a 数组元素不正确的引用是()。

A) a[p - a]

B) *(&a[i])

C) p[i]

D) a[10]

22. 阅读下列程序, 执行后的结果为()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int c[][4] = {1, 2, 3, 4, 5, 6, 7, 34, 213, 56, 62, 3, 23, 12, 34, 56};
```

```
printf("%x, %x\n", c[2][2], *(*(c+1)+1));
```

```
}
```

A) 3e, 6

B) 62, 5

C) 56, 56

D) 3E, 6

23. 阅读下列程序, 当运行函数时, 输入 asd af aa z67, 则输出为()。

```
#include <stdio.h>
```

```
#include <ctype.h>
```

```
#include <string.h>
```

```
void fun(char *str)
```

```
{
```

```
int i, j = 0;
```

```
for(i = 0; str[i] != '\0'; i++)
```

```
if(str[i] != ' ')
```

```
str[j++] = str[i];
```

```
str[j] = '\0';
```

```
}
```

```
void main()
```

```
{ char str[81];
```

```
int n;
```

```
printf("输入一字符串:");
```

```
gets(str);
```

```
puts(str);
```

```
fun(str);
```

```
printf("%s\n",str);
```

- A) asdafaaz67 B) asd af aa z67 C) asd D) z67

24. 假定以下程序经编译和连接后生成可执行文件 PROG. EXE, 如果在此可执行文件所在目录的 DOS 提示符下键入:

PROG ABCDEFGH IJKL <回车>, 则输出结果为()。

```
#include <stdio.h>
```

```
void main( int argc, char *argv[])
```

```
{ while( --argc > 0)
```

```
printf("%s", argv[argc]);
```

```
printf("\n");
```

- A) ABCDEFG B) IJHL C) ABCDEFGHIJKL D) IJKLABCDEFGH

25. 下面程序的输出结果是()。

```
#include <stdio.h>
```

```
#include <string.h>
```

```
void main( )
```

```
{
```

```
char *p1="abc", *p2="ABC", str[50]="xyz";
```

```
strcpy( str+2, strcat( p1, p2) );
```

```
printf("%s\n", str);
```

- A) xyzabcABC B) zabcABC C) xyabcABC D) yzabcABC

26. 有以下程序:

```
#include <stdio.h>
```

```
int fun( int (*s)[4], int n, int k)
```

```
{ int m, i;
```

```
m = s[0][k];
```

```
for(i=1; i<n; i++)
```

```
if(s[i][k] > m)
```

```
m = s[i][k];      //s[i][k]表示二维数组 a 的第 i 行第 k 列元素
```

```
return m;
```

```
void main( )
```

```
{ int a[4][4] = { {1,2,3,4}, {11,12,13,14}, {21,22,23,24}, {31,32,33,34} };
```

```
printf("%d\n", fun(a, 4, 0));
```

程序的运行结果是()。

- A) 4 B) 34 C) 31 D) 32

二、填空题

1. 以下程序的输出结果是_____。

```
#include <stdio.h>
int ast(int x, int y, int *cp, int *dp)
{ *cp = x + y; *dp = x - y; }
void main()
{ int a, b, c, d; a = 4; b = 3;
  ast(a, b, &c, &d);
  printf("%d %d\n", c, d); }
```

2. 若有定义: char ch;

- (1) 使指针 p 可以指向变量 ch 的定义语句是_____。
- (2) 使指针 p 可以指向变量 ch 的赋值语句是_____。
- (3) 通过指针 p 给变量 ch 读入字符的 scanf() 函数调用语句是_____。
- (4) 通过指针 p 给变量 ch 赋字符的语句是_____。
- (5) 通过指针 p 输出 ch 中字符的语句是_____。

3. 函数 void fun(float *sn, int n) 的功能是: 根据以下公式计算 s, 计算结果通过形参指针 sn 传回; n 通过形参传入, n 的值大于等于 0。请填空。

$$s = \sum_{i=0}^n (-1)^i / (2 \times i + 1)$$

```
void fun(float *sn, int n)
{ float s = 0.0, w, f = -1.0;
  int i = 0;
  for(i = 0; i <= n; i++)
  {
    f = _____ * f;
    w = f / (2 * i + 1);
    s + = w;
  }
  _____ = s; }
```

4. 若有图 8.35 (第 200 页) 所示 5 个连续的 int 类型的存储单元并赋值如图, 且 p 和 s 的基本类型皆为 int, p 已指向存储单元 a[1]。

- (1) 通过指针 p 给 s 赋值, 使其指向最后一个存储单元 a[4] 的语句是_____。
- (2) 用以移动指针 s, 使之指向中间的存储单元 a[2] 的表达式是_____。
- (3) 已知 k = 2, 指针 s 已指向存储单元 a[2], 表达式 *(s + k) 的值是_____。
- (4) 指针 s 已指向存储单元 a[2], 不移动指针 s, 通过 s 引用存储单元 a[3] 的表达式是_____。

(5) 指针 s 指向存储单元 a[2], p 指向存储单元 a[0], 表达式 s - p 的值是_____。

(6) 若 p 指向存储单元 a[0], 则以下语句的输出结果是_____。

```
for(i = 0; i < 5; i++) printf("%d", *(p + i));
printf("\n");
```

5. 以下程序从输入的 10 个字符串中找出最长的那个串, 请填空。

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#define N 10
```

```
void main()
```

```
{
```

```
    char str[N][81], *sp;
```

```
    int i;
```

```
    for(i=0; i<N; i++) gets(str[i]);
```

```
    sp = str[0];
```

```
    for(i=1; i<N; i++)
```

```
        if(strlen(sp) < strlen(str[i]))
```

```
            _____;
```

```
    printf("输出最长的那个串:\n%s\n", sp);
```

```
    printf("输出最长的那个串的长度:%d\n", strlen(sp));
```

```
}
```

6. 函数 my_cmp() 的功能是比较字符串 s 和 t 的大小, 当 s 等于 t 时返回 0, 否则返回 s 和 t 的第一个不同字符的 ASCII 码差值, 即 s > t 时返回正值, 当 s < t 时返回负值。请填空。

```
int my_cmp(char *s, char *t)
```

```
{
```

```
    while(*s == *t)
```

```
    { if(*s == '\0') return 0;
```

```
      ++s; ++t;
```

```
    }
```

```
    return _____;
```

```
}
```

三、编程题

1. 请编写函数, 其功能是对传送过来的两个浮点数求出和值与差值, 并通过形参传回调用函数。

2. 请编写函数, 对传送过来的 3 个数选出最大和最小数, 并通过形参传回调用函数。

3. 利用指针数组对 n 个字符串按字母顺序由小到大排序并输出。

____ 是第 i 个字符串的指针, [i] 是字符串数组的基址, 指向第 i 个字符串的起始地址。
____ 是第 i 个字符串的指针, [i] 是字符串数组的基址, 指向第 i 个字符串的起始地址。
____ 是第 i 个字符串的指针, [i] 是字符串数组的基址, 指向第 i 个字符串的起始地址。

____ 是第 i 个字符串的指针, [i] 是字符串数组的基址, 指向第 i 个字符串的起始地址。

____ 是第 i 个字符串的指针, [i] 是字符串数组的基址, 指向第 i 个字符串的起始地址。

____ 是第 i 个字符串的指针, [i] 是字符串数组的基址, 指向第 i 个字符串的起始地址。

____ 是第 i 个字符串的指针, [i] 是字符串数组的基址, 指向第 i 个字符串的起始地址。

第9章 编译预处理命令

编译预处理是指对 C 语言源程序进行编译处理之前对以“#”号开头的命令行进行的分析处理。这些命令包括宏定义、文件包含和条件编译。

9.1 宏定义

在前面各章中,已多次使用过以“#”号开头的预处理命令。如包含命令#include,宏定义命令#define等。

在 C 语言源程序中允许用一个标识符来表示一个字符串,称为“宏”。被定义为“宏”的标识符称为“宏名”。在编译预处理时,对程序中所有出现的“宏名”,都用宏定义中的字符串去代换,这称为“宏替换”或“宏展开”。

宏定义是由源程序中的宏定义命令完成的。宏替换是由预处理程序自动完成的。在 C 语言中,“宏”分为有参数和无参数两种。下面分别讨论这两种“宏”的定义和调用。

9.1.1 不带参数的宏定义

在程序中经常有一些常数是在全程序中不变化的。例如,处理数学问题的程序中用到的圆周率 π ,这个数 $PI=3.14159$ 是不变的,可以考虑用一个符号常量PI来处理它,不用每次在用到这个值时都写一次3.14159。

这种符号常量就是“宏”,它是不带参数的。这种宏的定义一般形式为:

```
#define 标识符 字符串
```

其中,“#”表示这是一条预处理命令(凡是以“#”开头的均为预处理命令);“define”为宏定义命令;“标识符”为所定义的宏名;“字符串”可以是常量、表达式、格式串等。

实际应用中,还经常对程序中反复使用的表达式进行宏定义。

【例 9.1】 利用不带参数的宏计算:(1)圆面积;(2)矩形面积;(3)三角形面积。

【参考代码】

```
#include <stdio.h>
#define PI 3.14
#define a 1.2
#define b 2.4
#define s2 a * b
#define x 1.2
#define y 2.4
#define s3 x * y / 2.0
void main()
{
    double r, s1;
```

```
printf("请输入圆的半径 r:");
scanf("%lf",&r);
s1 = PI * r * r;
printf("圆的面积 s1 = %lf\n",s1);
printf("矩形的面积 s2 = %lf\n",s2);
printf("三角形的面积 s3 = %lf\n",s3);
```

程序运行结果如图 9.1 所示。

【注意点】

(1) 在预定义中已经用到的符号不能在函数中再定义,如:

```
double s2,s3,a,b,x,y;
```

(2) 在不带参数的宏定义中像 a、b、x、y 必须先预定义后才能在后面的宏定义中使用,即宏定义可以嵌套,宏定义的字符串中可以使用已定义的宏名,在宏展开时由预处理程序层层代换,例如:

```
#define a 1.2
```

```
#define b 2.4
```

```
#define s2 a * b
```

(3) 在宏定义中,符号常量一般大写,本例中有大写也有小写。

(4) 编译器在编译之前把程序中出现的一切宏用其对应的表达式字符串来替代,而不是计算表达式的值后才替代,即宏定义是用宏名表示的一个字符串,在宏展开时又以该字符串取代宏,这只是一简单的代换,字符串可以含有任何字符,可以是常量,也可以是表达式,预处理程序对它不作任何检查。如有错误,只能在编译已被宏展开后的源程序时才发现。

例如:

```
printf("矩形的面积 s2 = %lf\n",s2);
```

被替换为:

```
printf("矩形的面积 s2 = %lf\n",1.2 * 2.4);
```

(5) 在 #define 语句中,宏与字符串之间用空格隔开,在表达式末尾没有分号,不然分号也会被认为是表达式的一部分而出现在程序中宏出现的位置。

(6) 如果在双引号中的字符串中出现宏名,则此字符串中的与宏名相同的字符是普通字符不会被替换。例如:

```
printf("矩形的面积 s2 = %lf\n",s2);
```

该语句中的第 1 个 s2 就不会被替换。另外对“输出格式”作宏定义,可以减少书写麻烦,但可读性差。

例如:

```
#include <stdio.h>
```

```
#define P printf
```

```
#define D "%d\n"
```

```
#define F "%f\n"
```

```
void main()
```

```
{
```

```
int a = 1, c = 2;
```

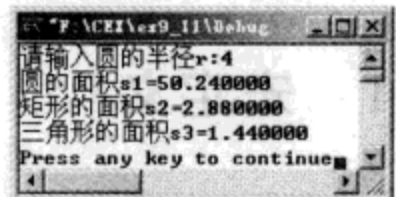


图 9.1 例 9.1 运行结果


```
float b = 1.2, d = 2.4;
```

```
P(D F,a,b);
```

```
P(D F,c,d);
```

9.1.2 带参数的宏定义

C语言允许宏带有参数。在宏定义中的参数称为形式参数,在宏调用中的参数称为实际参数。对带参数的宏,在调用中,不仅要宏展开,而且要用实参去代换形参。带参宏定义的一般形式为:

```
#define 宏名(形参表) 字符串
```

在字符串中含有各个形参。带参数的宏调用的一般形式为:

```
宏名(实参表);
```

例如:

```
#define S2(a,b) (a) * (b)
```

```
/* 宏定义 */
```

```
.....
```

```
printf("矩形的面积 S2 = %f\n", S2(1.2, 2.4)); /* 宏调用 */
```

```
.....
```

在宏调用时,用实参 1.2 和 2.4 去代替形参 a,b,经预处理宏展开后的语句为:

```
printf("矩形的面积 S2 = %f\n", 1.2 * 2.4);
```

【例 9.2】 利用带参数的宏定义计算:(1)圆面积;(2)矩形面积;(3)三角形面积。

【参考代码】

```
#include <stdio.h>
```

```
#define PI 3.14
```

```
#define S1(r) PI * (r) * (r)
```

```
#define S2(a,b) (a) * (b)
```

```
#define S3(x,y) (x) * (y) / 2.0
```

```
void main()
```

```
{
```

```
double r,a,b,x,y,s10,s20,s30;
```

```
printf("请输入圆的半径 r:");
```

```
scanf("%lf",&r);
```

```
printf("请输入矩形的长和宽 a,b:");
```

```
scanf("%lf%lf",&a,&b);
```

```
printf("请输入三角形的底和高 x,y:");
```

```
scanf("%lf%lf",&x,&y);
```

```
s10 = S1(r);
```

```
s20 = S2(a,b);
```

```
s30 = S3(x,y);
```

```
printf("圆的面积 = %f\n",s10);
```

```
printf("矩形的面积 = %f\n",s20);
```

```
printf("三角形的面积 = %f\n",s30);
```

```
}
```

程序运行结果如图 9.2 所示。

本例程序中进行了 3 个带参数的宏定义,用宏名 S1(r) 表示圆的面积 $PI * (r) * (r)$;用宏名 S2(a,b) 表示矩形的面积 $(a) * (b)$;用宏名 S3(x,y) 表示三角形的面积 $(x) * (y) / 2.0$ 。

在后面的语句:

```
s10 = S1(r);
s20 = S2(a,b);
s30 = S3(x,y);
```

为宏调用,这里的 r,a,b,x,y 都是实参,与宏定义的形参名称相同,当然也可以不同。

对于带参数的宏定义有以下问题需要说明。

(1) 带参宏定义中,宏名和形参表之间不能有空格出现。例如:

```
#define S2(a,b) (a) * (b)
```

写为:

```
#define S2(a,b) (a) * (b)
```

将被认为是无参宏定义,宏名 S2 代表字符串 (a,b) (a) * (b)。宏展开时,宏调用语句成为:

```
s20 = (a,b) (a) * (b)(a,b);
```

这显然是错误的。

(2) 在带参宏定义中,形式参数不分配内存单元,因此不必作类型定义。这与函数调用中的情况是不同的。在函数调用中,形参和实参是两个不同的量,各有自己的作用域,调用时要把实参值传递给形参,进行“值传递”。而在带参宏中,只是符号代换,不存在值传递的问题。

(3) 在宏定义中的形参是标识符,而宏调用中的实参可以是表达式。例如:

```
#define S2(a,b) (a) * (b)
```

不能写成:

```
#define S2(a,b) a * b
```

当宏调用 S2(a+2,3) 替换宏时,宏展开后为:

```
a+2*3
```

因此,这与实际不符。在宏定义中,字符串内的形参通常用括号括起来以避免出错。在上例的宏定义中(a) * (b)字符串中的 a 与 b 都用括号括起来,因此结果是正确的。

宏定义还可以解除。解除宏定义的一般形式为:

```
#undef 宏名
```

其中,#undef 是关键字;宏名是在此前定义过的。功能是解除前面已定义的宏,使之不再起作用,也就是说宏有它的作用域。

9.2 文件包含

文件包含是 C 预处理程序的另一个重要功能。

文件包含命令的一般形式为:

```
#include <文件名>
```

或

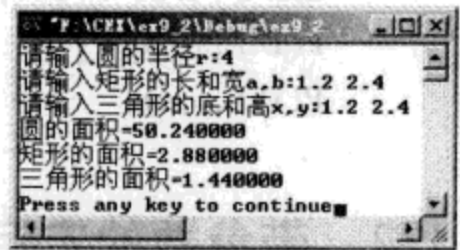


图 9.2 例 9.2 运行结果

#include "文件名"

【注意】

在前面已多次用此命令包含过库函数的头文件。例如:

```
#include <stdio.h>
#include <string.h>
```

在 Visual C++ 6.0 中,系统含有的头文件用两种形式都可以,自定义的头文件用#include <文件名>形式时第一次编译可能无法通过。如果不能通过请修改为#include "文件名"形式。文件包含命令的功能是把指定的文件插入到该命令行位置取代该命令行,从而把指定的文件和当前的源程序文件连成一个源文件。

关于文件包含命令的几点说明如下。

(1) 一个#include 命令只能指定一个被包含文件,若有多个文件要包含,则需用多个#include 命令。

(2) 被包含文件的扩展名可以不是.h 的文件。

(3) 文件包含允许嵌套,即在一个被包含的文件中又可以包含另一个文件。

【例 9.3】 利用文件包含计算:(1)圆面积;(2)矩形面积;(3)三角形面积。

【参考代码】

把如下代码存储到一个叫“area.h”的文件中,此文件要保存在与源程序相同的目录下。

```
#define PI 3.14
#define S1(r) PI * (r) * (r)
#define S2(a,b) (a) * (b)
#define S3(x,y) (x) * (y) / 2.0
```

程序用文件包含的形式包含此文件。参考代码如下:

```
#include <stdio.h>
#include "area.h"
void main()
{
    double r,a,b,x,y,s10,s20,s30;
    printf("请输入圆的半径 r:");
    scanf("%lf",&r);
    printf("请输入矩形的长和宽 a,b:");
    scanf("%lf%lf",&a,&b);
    printf("请输入三角形的底和高 x,y:");
    scanf("%lf%lf",&x,&y);
    s10 = S1(r);
    s20 = S2(a,b);
    s30 = S3(x,y);
    printf("圆的面积 = %lf\n",s10);
    printf("矩形的面积 = %lf\n",s20);
    printf("三角形的面积 = %lf\n",s30);
}
```

程序运行结果如图 9.2 所示。

【注意点】

#include "area. h"写成:#include <area. h>时,第一次编译可能产生如下错误:
fatal error C1083: Cannot open include file: 'area. h': No such file or directory

9.3 条件编译

预处理程序提供了条件编译的功能。可以按不同的条件去编译不同的程序部分,因而产生不同的目标代码文件。这对于程序的移植和调试是很有用的。条件编译有三种形式。

1. 第一种形式

#ifdef 标识符

程序段 1

#else

程序段 2

#endif

它的功能是,如果标识符已被 #define 命令定义过则对程序段 1 进行编译;否则对程序段 2 进行编译。如果没有程序段 2(它为空白),本格式中的 #else 可以没有,即可以写为:

#ifdef 标识符

程序段

#endif

【例 9.4】 利用条件编译形式 1 计算:(1)圆面积;(2)矩形面积;(3)三角形面积。

【参考代码】

```
#include <stdio. h>
#define PI 3. 14
#define S1(r) PI * (r) * (r)
// #define S2(a,b) (a) * (b)
// #define S3(x,y) (x) * (y) / 2. 0
void main()
{
    #ifdef S1
    { double r,s10;
      printf("请输入圆的半径 r:");
      scanf("%lf",&r);
      s10 = S1(r);
      printf("圆的面积 = %lf\n",s10);
    }
    #endif
    #ifdef S2
    //因 S2(a,b) 没有定义,所以不能输出矩形的面积
    { double a,b,s20;
      printf("请输入矩形的长和宽 a,b:");
      scanf("%lf%lf",&a,&b);
      s20 = S2(a,b);
```

```

printf("矩形的面积 = %lf\n",s20);
#endif
#ifdef S3 //因 S3(a,b) 没有定义,所以不能输出三角形的面积
{
double x,y,s30;
printf("请输入三角形的底和高 x,y:");
scanf("%lf%lf",&x,&y);
s30 = S3(x,y);
printf("三角形的面积 = %lf\n",s30);
}
#endif

```

程序运行结果如图 9.3 所示。

【注意点】

(1) 不带参数与带参数的宏定义在条件编译格式 1 中的使用

格式都一样:

#ifdef 宏名

带参数的宏中的参数不能含有,请读者上机操作。

(2) 程序中#ifdef #else #endif 可以用嵌套方式。本例程序片断可为:

```

#ifdef S1
{.....}
#else ifdef S2
{.....}
#else ifdef S3
{.....}
#endif
#endif
#endif

```

2. 第二种形式

#ifndef 标识符

程序段 1

#else

程序段 2

#endif

与第一种形式的区别是将“ifdef”改为“ifndef”。它的功能是,如果标识符未被#define 命令定义过则对程序段 1 进行编译,否则对程序段 2 进行编译。这与第一种形式的功能正相反。

3. 第三种形式

#if 常量表达式

程序段 1

#else

程序段 2

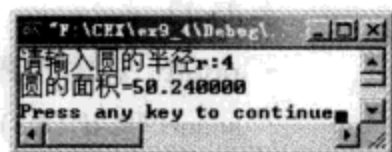


图 9.3 例 9.4 运行结果

```
#endif
```

它的功能是,如常量表达式的值为真(非0),则对程序段1进行编译,否则对程序段2进行编译。因此可以使程序在不同条件下,完成不同的功能。

【例 9.5】 利用条件编译形式3 计算:(1)圆面积;(2)矩形面积;(3)三角形面积。

```
#include <stdio.h>
#define CHOICE1 0
#define CHOICE2 1
#define CHOICE3 0
void main()
{
    #if CHOICE1
    { double r;
      printf("请输入圆的半径 r:");
      scanf("%lf",&r);
      printf("圆的面积 = %lf\n",3.14159 * r * r);
    }
    #endif
    #if CHOICE2
    { double a,b;
      printf("请输入矩形的长和宽 a,b:");
      scanf("%lf%lf",&a,&b);
      printf("矩形的面积 = %lf\n",a * b);
    }
    #endif
    #if CHOICE3
    { double x,y;
      printf("请输入三角形的底和高 x,y:");
      scanf("%lf%lf",&x,&y);
      printf("三角形的面积 = %lf\n",x * y/2);
    }
    #endif
}
```

程序运行结果如图 9.4 所示。

本例中采用了第三种形式的条件编译。在程序定义了三个宏定义中,定义 CHOICE2 为 1,因此在条件编译时,常量表达式的值为真,故计算并输出矩形的面积。

用条件编译,根据条件只编译其中的程序段 1 或程序段 2,生成的目标程序较短。

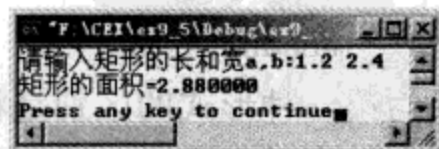


图 9.4 例 9.5 运行结果

本章小结

(1) 预处理功能是 C 语言特有的功能,它是在对源程序正式编译前由预处理程序自动完成的。

(2) 宏定义是用一个标识符来表示一个字符串,这个字符串可以是常量、变量或表达式。在宏展开中将用该字符串代换宏名。宏定义可以带有参数,宏调用时是以实参代换形参,而不是“值传送”。为了避免宏代换时发生错误,宏定义中的字符串应加括号,即字符串中出现的形式参数两边应加括号。

(3) 文件包含可用来把多个源文件连接成一个源文件进行编译,结果将生成一个目标文件。

(4) 条件编译允许只编译源程序中满足条件的程序段,使生成的目标程序较短,从而减少了内存的开销,并提高了程序的效率。

习题九

一、选择题

1. 以下程序的输出结果是()。

```
#include <stdio.h>
#define MIN(x,y) (x)<(y)?(x):(y)
void main()
{
    int i,j,k;
    i=10;
    j=15;
    k=10*MIN(i,j);
    printf("%d\n",k);
}
```

A) 15 B) 100 C) 10 D) 150

2. 阅读下列程序段,程序的输出结果为()。

```
#include <stdio.h>
#define M(X,Y) (X)*(Y)
#define N(X,Y) (X)/(Y)
void main()
{
    int a=5,b=6,c=8,k;
    k=N(M(a,b),c);
    printf("%d\n",k);
}
```

A) 3 B) 5 C) 6 D) 8

3. 以下程序中的 for 循环执行的次数是()。

```
#include <stdio.h>
```

```
#define N 2
```

```
#define M N + 1
```

```
#define NUM (M + 1) * M / 2
```

```
void main()
```

```
{
```

```
    int i;
```

```
    for(i=0; i<=NUM; i++)
```

```
        printf("%d\n", i);
```

```
}
```

A)5

B)6

C)8

D)9

4. 若要求定义具有 10 个 int 型元素的一维数组 a, 则以下定义语句中错误的是()。

A) #define N 10

B) #define n 5

C) int a[5+5];

D) int n=10, a[n];

```
int a[N];
```

```
int a[2 * n];
```

5. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
#define FU(y) 2.84 + y
```

```
#define PR(a) printf("%d", (int)(a))
```

```
#define PRINT1(a) PR(a); putchar('\n')
```

```
void main()
```

```
{
```

```
    int x=2;
```

```
    PRINT1(FU(5) * x);
```

```
}
```

A)11

B)12

C)13

D)15

6. 以下叙述中错误的是()。

A) C 程序中的 #include 和 #define 行均不是 C 语句

B) 除逗号运算符外, 赋值运算符的优先级最低

C) C 程序中, j++; 是赋值语句

D) C 程序中, +、-、*、/、% 号是算术运算符, 可用于整型和实型数的运算

7. 以下叙述中正确的是()。

A) 用 #include 包含的头文件的后缀不可以是“.a”

B) 若一些源程序中包含某个头文件, 当该头文件有错时, 只需对该头文件进行修改, 包含此头文件所有源程序不必重新进行编译

C) 宏命令行可以看作是一行 C 语句

D) C 编译中的预处理是在编译之前进行的

8. 有以下程序:

```
#define P 3
```

```
#define F(int x) { return (P * x * x); }
```

```
void main()
```

```
{ printf("%d", F(3+5)); }
```

程序运行后的输出结果是 ()。

A) 192 B) 29 C) 25 D) 编译出错

9. 以下叙述中正确的是 ()。

- A) 预处理命令必须位于源文件的开头
- B) 在源文件的一行上可以有多条预处理命令
- C) 宏名必须用大写字母表示
- D) 宏替换不占用程序的运行时间

二、填空题

1. 以下程序的输出结果是_____。

```
#include <stdio.h>
```

```
#define PR(array) printf("%d", array)
```

```
void main()
```

```
{ int j, a[] = {1, 3, 5, 7, 9, 11, 13, 15}, *p = a + 5;
```

```
  for (j = 3; j > 0; j--)
```

```
  switch(j)
```

```
  { case 1:
```

```
    case 2: PR(*p++); break;
```

```
    case 3: PR(*(--p));
```

```
  }
```

2. 现有两个 C 程序文件 T18.c 和 myfun.h 同时在 TC 系统目录(文件夹)下, 其中 T18.c 文件如下:

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include "myfun.h"
```

```
void main()
```

```
{ fun();
```

```
  printf("\n");
```

```
}
```

myfun.h 文件如下:

```
void fun()
```

```
{ char s[80], c;
```

```
  int n = 0;
```

```
  while((c = getchar()) != '\n')
```

```
  s[n++] = c;
```

```
  n--;
```

```
  while(n >= 0)
```

```
  printf("%c", s[n--]);
```



当编译连接通过后,运行程序 T18.exe 时,输入 Thank! 则输出的结果是:_____。

三、编程题

1. 请写出一个宏定义 MYALPHA(C),用以判断 C 是否是字母字符,若是,得 1,否则得 0。
2. 定义一个宏 LEAP_YEAR(y),判断 y 是否为闰年。
3. 定义一个宏,能找出输入三个数的最大值。

```
void main()
{
    int a[] = {1, 3, 2, 7, 9, 11, 13, 15};
    for (j = 3; j < 0; j--)
    {
        switch(j)
        {
            case 1: PR(*p++); break;
            case 3: PR(*(--p));
        }
    }
}
```

2. 现有两个 C 语言文件 T18.c 和 myfun.h 同时在 VC 系统目录下,其中 T18.c

文件如下:

```
#include <stdio.h>
#include <string.h>
#include "myfun.h"

void main()
{
    char a[80];
    int n = 0;
    while((c = getchar()) != '\n')
    {
        n++;
        a[n-1] = c;
    }
    while(n > 0)
    {
        printf("%c", a[n-1]);
    }
}
```

第10章 结构与共用

在C语言中,一组类型相关的数据,用数组来表示。但有时可能会遇到一组类型不同的相关数据,这时用数组就无法表示。例如,某学校要建立学生基本情况的档案,档案中包含有学生的学号、姓名、性别、年龄、入学日期、身份证号及班级编号等7项信息,用函数实现以下功能:

(1)将学号从小到大排列,相应的学生姓名同时调整;

(2)输入学生学号,用一定算法找到相应的学生姓名,并输出该学生的全部信息。

由于每个学生有7项信息,且信息的类型不相同,如年龄是整型,学号可以是整型或字符型,姓名只能是字符型等。在处理此问题时,显然不能用数组来处理学生的这些信息。这些信息的类型虽然不同,但它们之间是有内在联系的。如果将学生的这些信息独立存放在不同的变量或数组中,就很难看出它们之间的联系。

为此,在高级语言中,大多都会引进用户自定义数据类型的概念,以方便程序设计者构造新的数据类型。在C语言中,这种类型称结构。

10.1 结构与结构变量

10.1.1 结构的定义

在C语言中,结构是数据类型可以不同的变量的集合。组成结构的变量称为结构元素或结构成员。

结构定义的一般形式为:

```
struct 结构名
{
    类型说明符 成员名1;
    类型说明符 成员名2;
    .....
    类型说明符 成员名n;
};
```

其中,struct 是关键字,每个结构都可以含有多个不同类型的成员,结构中的成员也称结构中的元素,结构定义以分号“;”结束。

成员名的命名应符合标识符的书写规定。例如:

```
struct student
{
    int id;           //学号
    char name[9];     //姓名
    char sex;         //性别
```

```
int age;           //年龄
};
```

在这个结构定义中,结构名为 student,该结构由 4 个成员组成。第一个成员为整型变量 id;第二个成员为字符数组 name;第三个成员为字符变量 sex;第四个成员为整型变量 age。

结构类型说明:

struct student 是一个新的数据类型,这个类型类似于其他基本数据类型,以后就可以用这个新的数据类型来定义变量。一定要理解,student 不是变量,不能直接使用,但能用来定义变量。

结构类型的定义只是给出了该结构的组成情况,标志着这种类型的结构“模式”已存在,编译程序并没有因此而分配任何存储空间,真正占有存储空间的是具有相应结构类型的变量。

10.1.2 结构变量的定义

定义结构变量有以下三种方法。以上面定义的 student 为例来定义结构变量。

1. 先定义结构,再定义结构变量
如:

```
struct student
```

```
{
    int id;
    char name[9];
    char sex;
    int age;
};
```

```
struct student stu1, stu2;
```

定义两个变量 stu1 和 stu2 为 student 结构类型。也可以用一个符号常量来表示一个结构类型。

例如:

```
#define STUDENT struct student
```

```
STUDENT
```

```
{
    int id;
    char name[9];
    char sex;
    int age;
};
```

```
STUDENT stu1, stu2;
```

2. 在定义结构类型的同时定义结构变量

例如:

```
struct student
```

```
{
    int id;
    char name[9];
    char sex;
```



```

char sex;
int age;
} stu1, stu2;

```

这种形式的定义一般形式为:

```

struct 结构名
{
    成员表列;
    变量名表列;
}

```

3. 直接定义结构变量

例如:

```

struct
{
    int id;
    char name[9];
    char sex;
    int age;
} stu1, stu2;

```

这种形式的定义一般形式为:

```

struct
{
    成员表列;
    变量名表列;
}

```

第三种方法与第二种方法的区别在于第三种方法省去了结构名,而直接给出结构变量。三种方法中定义的 stu1 和 stu2 变量都具有如图 10.1 所示的结构。

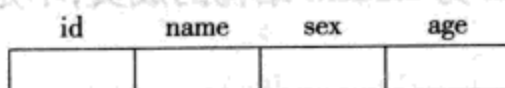


图 10.1 结构模式

定义 stu1 和 stu2 变量为 student 类型后,即可向这两个变量中的各个成员赋值。在上述 student 结构定义中,所有的成员都是基本数据类型或数组类型。

成员也可以是一个结构,即构成了嵌套的结构。例如,如图 10.2 给出了另一个结构。

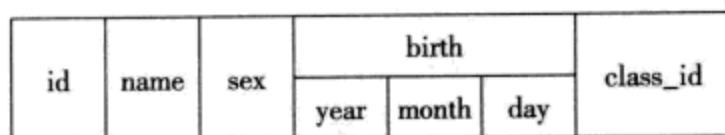


图 10.2 结构的嵌套

按图可给出以下结构定义:

```

struct birthday
{
    int year;
    int month;
    int day;
}

```

```

};
struct
{
    int id;
    char name[20];
    char sex;
    struct birthday birth;
    char class_id[6];
} stu1, stu2;

```

首先定义一个结构 birthday, 由 year(年)、month(月)、day(日)三个成员组成。在定义变量 stu1 和 stu2 时, 其中的成员 birth 被定义为 birthday 结构类型。成员名可与程序中其他变量同名, 互不干扰。

10.1.3 结构变量的引用与赋值

1. 结构变量的引用

一般有两种方式:

结构变量名. 成员名(用“.”号来访问结构变量的成员)

结构变量名 → 成员名(用“→”号来访问结构变量的成员)

例如:

stu1.id //第1种方式

stu1 → sex //第2种方式

如果成员本身又是一个结构, 则必须逐级找到最低级的成员才能使用。

例如:

stu1.birth.month (birth 为 student 结构的成员, 不是结构名)

2. 结构变量的赋值

结构变量的赋值就是给各成员赋值。可用输入语句或赋值语句来完成。

【例 10.1】编写程序, 利用结构变量存储一位学生的部分信息和 3 门课程成绩, 要求数据通过键盘输入, 并输出。

【参考代码】

```

#include <stdio.h>
struct student
{
    int id;
    char name[20];
    char sex;
    int age;
    float score[3];
};
void main()
{
    struct student stu;

```



图 10.1 学生记录结构

```

int i;
printf("请输入一学生的学号、姓名、性别、年龄:\n");
scanf("%d%s",&stu.id,stu.name);
getchar();           //接收输入时的回车符
stu.sex = getchar();
scanf("%d",&stu.age); //输入年龄
printf("请输入该学生三门课程成绩:\n");
for(i=0;i<3;i++)
scanf("%f",&stu.score[i]);
printf("输入的信息如下:\n");
printf("学号:%d\n 姓名:%s\n 性别:%c\n 年龄:%d\n",
stu.id,stu.name,stu.sex,stu.age);
printf("成绩:");
for(i=0;i<3;i++)
printf("%5.1f",stu.score[i]);
printf("\n");
}

```

程序运行结果如图 10.3 所示。

【注意点】

(1) 语句“scanf("%d%s",&stu.id,stu.name);”不能书写成：“scanf("%d%s",&stu.id,&stu.name);”因为 name 是字符数组,它代表数组的首地址,已经是地址。

(2) 在上面提到的语句中,输入的字符串不能含有空格,如果用 gets(stu.name) 输入时要有回车符表示字符串结束。

(3) 第一个“getchar();”语句,用于接收回车符。

(4) 语句“scanf("%f",&stu.score[i]);”不能写成：“scanf("%f",&stu.score);”

(5) 注意定义 student 结构时用“{};”形式,括号后的“;”不能少。

本例是简单结构的数据输入与输出,请读者上机体会。

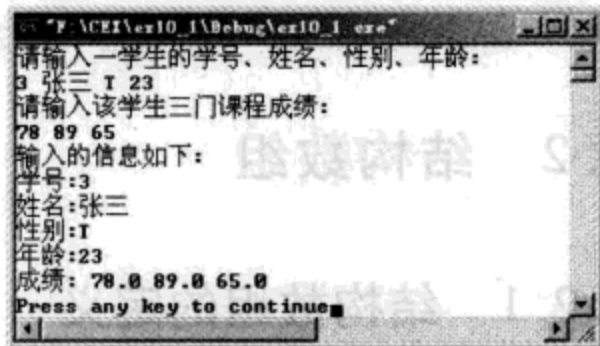


图 10.3 例 10.1 运行结果

10.1.4 结构变量的初始化

与其他类型变量一样,对结构变量也可以在定义时进行初始化。

【例 10.2】 对结构变量的初始化。

【参考代码】

```

#include <stdio.h>
void main()
{
    struct student          //定义结构
    {
        int id;

```

```
char name[20];
char sex;
int age;
float score[3];
} stu = {3, "张三", 'T', 23, 78, 89, 65}; //初始化结构变量, "T"表示性别为"男"
int i;
printf("输出的信息如下:\n"); //输出结构变量的信息
printf("学号:%d\n 姓名:%s\n 性别:%c\n 年龄:%d\n",
stu.id, stu.name, stu.sex, stu.age);
printf("成绩:");
for(i=0; i<3; i++)
printf("%5.1f", stu.score[i]);
printf("\n");
}
```

【注意点】

(1) 结构 student 可以在主函数外定义,也可以在主函数内定义。

(2) 结构变量与全局变量、局部变量的作用范围类似,可以定义为全局变量,也可定义为局部变量。如果结构定义在主函数外部,而结构变量在主函数内部,则应如下初始化:

```
struct student stu = {3, "张三", 'T', 23, 78, 89, 65};
```

10.2 结构数组

10.2.1 结构数组的定义

结构是一种构造类型,它就像普通数据类型一样,可以用结构类型来定义数据,也就是说数组的元素也可以是结构类型的,因此可以构成结构型数组。结构数组的每一个元素都是具有相同结构类型的下标结构变量。在实际应用中,经常用结构数组来表示具有相同数据结构的一个群体。如一个班的学生档案、一个车间职工的工资表等。

方法与结构变量相似,只需说明它为数组类型即可。例如:

```
struct student
{
    int id;
    char name[20];
    char sex;
    int age;
    float score[3];
} stu[3];
```

定义了一个结构数组 stu,共有 3 个元素,stu[0] ~ stu[2]。每个数组元素都具有 struct student 的结构形式。

10.2.2 结构数组的初始化

对结构数组也可以作初始化。例如:

```

struct student
{
    int id;
    char name[20];
    char sex;
    int age;
    float score[3];
    stu[3] = { {3, "张三", 'T', 23, 78, 89, 65}, // "T"表示性别为"男", "F"表示性别为"女"
               {4, "李四", 'F', 22, 73, 54, 75},
               {5, "王五", 'T', 19, 45, 88, 69} };

```

当对全部元素初始化赋值时,也可不给出数组长度。

10.2.3 结构数组应用

【例 10.3】 从键盘上输入 3 名学生的信息,计算每名学生三门课程的总分以及每门课程的平均成绩。

【参考代码】

```
#include <stdio.h>
```

```
struct student
```

```

{
    int id;
    char name[20];
    char sex;
    int age;
    float score[3];
    float sum;
};

```

```
void main()
```

```
{
    struct student stu[3];
```

```
    int i, j;
```

```
    float aver[3] = {0, 0, 0};
```

```
    printf("请输入学生的学号、姓名、性别、年龄、成绩 1、成绩 2、成绩 3:\n");
```

```
    for(i = 0; i < 3; i++)
```

```
    {
        scanf("%d%s", &stu[i].id, &stu[i].name);
```

```
        getchar();
```

//接收输入时用于姓名和性别之间的回车符

```
        stu[i].sex = getchar();
```

```
        scanf("%d", &stu[i].age);
```

//输入年龄

```
        for(j = 0; j < 3; j++)
```

```
            scanf("%f", &stu[i].score[j]);
```

```
        for(stu[i].sum = 0, j = 0; j < 3; j++)
```

```
            stu[i].sum += stu[i].score[j];
```



```

    for(j=0;j<3;j++)
        aver[j] = aver[j] + stu[i].score[j];
}

printf("输出的信息如下:\n");
printf("学号 姓名 性别 年龄 成绩1 成绩2 成绩3 总分\n");
for(i=0;i<3;i++)
{
    printf("%3d %8s %c %5d ",stu[i].id,stu[i].name,stu[i].sex,stu[i].age);
    for(j=0;j<3;j++)
        printf("%8.1f",stu[i].score[j]);
    printf("%8.1f",stu[i].sum);
    printf("\n");
}

printf("学生成绩平均分:\n");
printf("成绩1 成绩2 成绩3:\n");
for(j=0;j<3;j++)
    printf("%5.1f ",aver[j]/3);
printf("\n");
}

```

程序运行结果如图 10.4 所示。

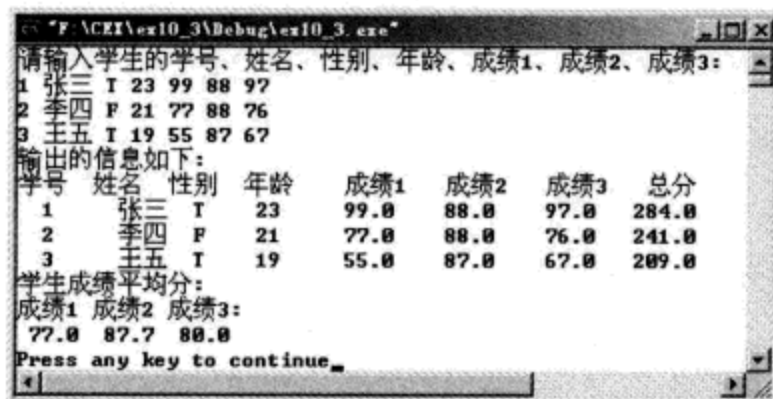


图 10.4 例 10.3 运行结果

10.3 指向结构的指针变量

10.3.1 指向结构变量的指针

当一个指针变量用来指向一个结构变量时,称之为结构指针变量。结构指针变量中的值是所指向的结构变量的首地址。通过结构指针即可访问该结构变量,这与数组指针和函数指针的情况是相同的。

结构指针变量定义的一般形式为:

struct 结构名 * 结构指针变量名;

例如,在前面的例题中定义了 student 这个结构,如要定义一个指向 student 结构的指针变量 p,可写为:


```
struct student *p;
```

当然也可在定义 stu 结构时同时定义 p。与前面讨论的各类指针变量相同,结构指针变量也必须要先赋值后使用。赋值是把结构变量的首地址赋予该指针变量,不能把结构名赋予该指针变量。如果 stu 被定义为 student 类型的变量,则“p = &stu;”是正确的,而“p = &student;”是错误的。

结构名和结构变量是两个不同的概念,不能混淆。结构名只能表示一个结构形式,编译系统并不对它分配内存空间。只有当某变量被定义为这种类型的结构时,才对该变量分配存储空间。因此,上面“&student”这种写法是错误的,不可能去取一个结构名的首地址。有了结构指针变量,就能更方便地访问结构变量的各个成员。

其访问的一般形式为:

(* 结构指针变量). 成员名

或

结构指针变量 -> 成员名

例如:

(* p). id

或者:

p -> id

应该注意 (*p) 两侧的括号不可少,因为成员符“.”的优先级高于“*”。如去掉括号写作 *p.id 则等效于 *(p.id), 意义就完全不对了。

下面通过例子来说明结构指针变量的具体定义和使用方法。

【例 10.4】 结构指针变量的使用。

```
#include <stdio.h>

void main()
{
    struct student          //定义结构
    {
        int id;
        char *name;         //char name[20]
        char sex;
        int age;
        float score[3];
    } stu = {3, "张三", 'T', 23, 78, 89, 65}, *p;

    //初始化结构变量,定义一个结构指针变量

    int i;
    p = &stu;               //指向结构变量
    printf("输出的信息如下:\n"); //输出结构变量的信息
    printf("学号:%d\n 姓名:%s\n 性别:%c\n 年龄:%d\n",
           stu.id, stu.name, stu.sex, stu.age);
    p -> id, p -> name, p -> sex, p -> age);
    printf("成绩:");
    for(i=0; i<3; i++)
        printf("%5.1f", p -> score[i]);
}
```

```
printf("\n");
```

【程序说明】

本例程序定义了一个结构 student, 定义了 student 类型的结构变量 stu 并作了初始化, 还定义了一个指向 student 类型的结构的指针变量 p。在 main() 函数中, p 被赋予 stu 的地址, 因此 p 指向 stu。然后用 printf() 语句输出 stu 的各个成员值。如果本例采用访问结构成员的三种形式, 从运行结果可以看出:

结构变量. 成员名

(* 结构指针变量). 成员名

结构指针变量 -> 成员名

这三种用于表示结构成员的形式是完全等效的。程序运行结果如图 10.5 所示。

10.3.2 指向结构数组的指针

指针变量可以指向一个结构数组, 这时结构指针变量的值是整个结构数组的首地址。结构指针变量也可指向结构数组的一个元素, 这时结构指针变量的值是该结构数组元素的首地址。

设 p 为指向结构数组的指针变量, 则 p 也指向该结构数组的 0 号元素, p + 1 指向 1 号元素, p + i 则指向 i 号元素。这与普通数组的情况是一致的。

【例 10.5】用指针变量输出结构数组。要求实现例 10.3 功能。

```
#include <stdio.h>
```

```
struct student
```

```
{
```

```
int id;
```

```
char name[20];
```

```
char sex;
```

```
int age;
```

```
float score[3];
```

```
float sum;
```

```
};
```

```
void main()
```

```
{
```

```
struct student stu[3], *p;
```

```
int j;
```

```
float aver[3] = {0, 0, 0};
```

```
printf("请输入学生的学号、姓名、性别、年龄、成绩 1、成绩 2、成绩 3:\n");
```

```
for(p = stu; p < stu + 3; p++)
```

```
{
```

```
scanf("%d%s", &p->id, p->name);
```

```
getchar();
```

```
//接收输入时用于姓名和性别之间的回车符
```

```
p->sex = getchar();
```

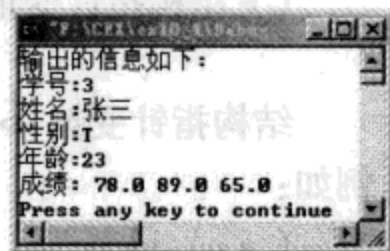


图 10.5 例 10.4 运行结果

```

scanf("%d",&p->age);          //输入年龄
for(j=0;j<3;j++)
    scanf("%f",&p->score[j]);
for(p->sum=0,j=0;j<3;j++)
    p->sum+=p->score[j];
for(j=0;j<3;j++)
    aver[j]=aver[j]+p->score[j];
}

printf("输出的信息如下:\n");
printf("学号 姓名 性别 年龄 成绩1 成绩2 成绩3 总分\n");
for(p=stu;p<stu+3;p++)
{
    printf("%3d %8s %c %5d ",p->id,p->name,p->sex,p->age);
    for(j=0;j<3;j++)
        printf("%8.1f",p->score[j]);
    printf("%8.1f",p->sum);
    printf("\n");
}

printf("学生成绩平均分:\n");
printf("成绩1 成绩2 成绩3:\n");
for(j=0;j<3;j++)
    printf("%5.1f",aver[j]/3);
printf("\n");
}

```

程序运行结果如图 10.6 所示。

```

C:\ACEI\ex10_5\Debug\ex10_5.exe
请输入学生的学号、姓名、性别、年龄、成绩1、成绩2、成绩3:
1 gggg I 23 77 88 99
2 ffff F 22 88 99 77
3 ssss I 21 88 77 66
输出的信息如下:
学号 姓名 性别 年龄 成绩1 成绩2 成绩3 总分
1 gggg I 23 77.0 88.0 99.0 264.0
2 ffff F 22 88.0 99.0 77.0 264.0
3 ssss I 21 88.0 77.0 66.0 231.0
学生成绩平均分:
成绩1 成绩2 成绩3:
84.3 88.0 80.7
Press any key to continue.

```

图 10.6 例 10.5 运行结果

【注意点】

“p = stu”表示把结构数组 stu[3]的首地址赋给 p,也可用“p = &stu[0]”。但不能出现 p = &stu[1].sex,因为它表示结构指针变量指向结构的一个成员,这是不允许的。

10.3.3 指向结构变量的指针变量作函数参数

C 语言中允许指向结构变量的指针变量作为函数参数传递。

【例 10.6】用指向结构变量的指针变量作函数参数编程,要求实现例 10.4 的功能。

【参考代码】

```

#include <stdio.h>

struct student
{
    int id;
    char name;
    char sex;
    int age;
    float score[3];
};

void fun(struct student *ps)
{
    int i;
    printf("输出的信息如下:\n"); //输出结构变量的信息
    printf("学号:%d\n 姓名:%s\n 性别:%c\n 年龄:%d\n",
        ps->id, ps->name, ps->sex, ps->age);
    printf("成绩:");
    for(i=0; i<3; i++)
        printf("%5.1f", ps->score[i]);
    printf("\n");
}

void main()
{
    struct student stu = {3, "张三", 'T', 23, 78, 89, 65}, *p;
    //初始化结构变量,定义一个结构指针变量
    p = &stu; //指向结构变量
    fun(p);
}

```

程序运行结果如图 10.5 所示。

【程序说明】

本程序中定义了函数 fun(), 其形参为结构指针变量 ps。stu 被定义为局部结构变量, 并被初始化。在 main() 函数中定义了结构指针变量 p, 并把 stu 的首地址赋予它, 使 p 指向 stu 结构变量。然后用 p 作实参调用函数 fun(), 在函数 fun() 中完成学生的信息输出。

10.3.4 指向结构数组的指针变量作函数参数

在 ANSI C 标准中允许用结构数组作函数参数进行整体传送。但是这种传送要将全部成员逐个传送, 特别是成员为数组时将会使传送的时间和空间开销很大, 严重地降低了程序的效率。因此, 最好的办法就是使用指针, 即用指针变量作函数参数进行传送。这时由实参传向形参的只是地址, 从而减少了时间和空间的开销。

【例 10.7】 用指向结构数组的指针变量作函数参数编程, 要求实现例 10.3 的输出功能。

【参考代码】

```

#include <stdio.h>

struct student
{
    int id;
    char name[20];
    char sex;
    int age;
    float score[3];
    float sum;
} stu[3]; //stu 为全局结构数组

void fun(struct student *p)
{
    int j;
    float aver[3] = {0,0,0};
    for(p = stu; p < stu + 3; p++)
    {
        for(p->sum = 0, j = 0; j < 3; j++)
            p->sum += p->score[j];
        for(j = 0; j < 3; j++)
            aver[j] = aver[j] + p->score[j];
    }

    printf("输出的信息如下:\n");
    printf("学号 姓名 性别 年龄 成绩1 成绩2 成绩3 总分\n");
    for(p = stu; p < stu + 3; p++)
    {
        printf("%3d %8s %c %5d", p->id, p->name, p->sex, p->age);
        for(j = 0; j < 3; j++)
            printf("%8.1f", p->score[j]);
        printf("%8.1f", p->sum);
        printf("\n");
    }

    printf("学生成绩平均分:\n");
    printf("成绩1 成绩2 成绩3:\n");
    for(j = 0; j < 3; j++)
        printf("%5.1f", aver[j]/3);
    printf("\n");
}

void main()
{
    struct student *p;
    int j;
    float aver[3] = {0,0,0};

```

```

printf("请输入学生的学号、姓名、性别、年龄、成绩1、成绩2、成绩3:\n");
for(p = stu; p < stu + 3; p++)
{
    scanf("%d%s", &p->id, p->name);
    getchar(); //接收输入时用于姓名和性别之间的回车符
    p->sex = getchar();
    scanf("%d", &p->age); //输入年龄
    for(j = 0; j < 3; j++)
        scanf("%f", &p->score[j]);
}
p = stu; //p 指向 stu 结构数组首地址
fun(p);

```

程序运行结果如图 10.6 所示。

【程序说明】

本程序中定义了函数 fun(), 其形参为结构指针变量 p。stu 被定义为外部结构数组, 因此在整个源程序中有效。在 main() 函数中定义了结构指针变量 p, 语句“p = stu;”是把 stu 的首地址赋予它, 使 p 指向 stu 数组。然后用 p 作实参调用函数 fun(), 在函数 fun() 中完成计算每名学生的三门总分和每门课程的平均成绩。

10.4 类型定义符 typedef

C 语言允许用户使用 typedef 定义类型说明符。新的类型标识符可当作原标识符使用。

类型定义符的一般形式为:

typedef 类型说明符 标识符;

其中,“类型说明符”是已有类型说明符,如 int、float、char 等;“标识符”是用户定义的新类型标识符,新的标识符一般用大写表示,以便于区别。

例如:

```
typedef int INTEGER;
```

该语句把 INTEGER 说明成了一个 int 类型的类型名,这样就能用新定义的类型标识符 INTEGER 去定义变量和数组了。例如:

```
INTEGER a, b;
```

等效于:

```
int a, b;
```

用 typedef 定义数组、指针、结构等类型将带来很大的方便,不仅使程序书写简单而且使意义更为明确,因而增强了可读性。例如:

```
typedef char STR[80];
```

表示 STR 是字符数组类型,数组长度为 80。然后可用 STR 定义变量,如:

```
STR s1, s2, s3, s4;
```

完全等效于:

```
char s1[80], s2[80], s3[80], s4[80];
```


又如:

```
typedef struct student
```

```
    char name[20];
```

```
    int age;
```

```
    char sex;
```

```
}; STU;
```

定义 STU 表示 student 的结构类型,然后可用 STU 来定义结构变量:

```
STU stu1, stu2;
```

再如:

```
typedef float * PFLOAT;
```

```
PFLOAT p1, p2;
```

等价于:

```
float * p1, * p2;
```

typedef 的说明:

- (1) 用 typedef 可以将已有的各种类型定义成新的类型说明符,但不能直接定义变量;
- (2) typedef 只是对已有的类型说明符增加一个新类型说明符来替换它,并不能创造新的数据类型;

(3) 用 typedef 定义一个新类型说明符后,还可以再用 typedef 将新类型说明符定义成另一个新的类型说明符。

例如:

```
typedef float FREAL;
```

```
typedef FREAL FR;
```

10.5 结构与链表

10.5.1 动态存储结构

在学习数组时,数组的长度往往是先定义好,在整个程序中是固定不变的。C 语言不允许使用动态数组。

例如:

```
int n;
```

```
scanf("%d", &n);
```

```
int a[n];
```

这种定义数组的方法是错误的。

在实际应用中,往往会发生这种情况,即所需要的内存空间取决于实际输入的数据,而无法事先确定。这种问题用数组的方法很难解决。为了解决这个问题, C 语言提供了一些内存管理函数,这些内存管理函数可以按照需要动态地分配内存空间,也可以把不再使用的空间释放待用,从而可以有效地、合理地使用内存空间。常用的内存管理函数有如下 3 个。

1. 分配内存空间函数 malloc()

malloc() 函数的一般形式为:

(类型说明符 *) malloc(size);

malloc() 函数说明如下。

(1) 该函数可在内存的动态存储区分配一块“size”字节的连续区域。函数的返回值为该区域的首地址。

(2) “类型说明符”表示把该区域用于何种数据类型。(类型说明符 *) 表示把返回值强制转换为该类型指针。“size”为无符号数。

例如:

```
char * p;
```

```
p = (char *) malloc(20);
```

表示分配 20 个字节的内存空间, 并把其首地址赋给指向字符的指针变量 p。

2. 分配内存空间函数 calloc()

calloc() 也用于分配内存空间。其一般形式为:

(类型说明符 *) calloc(n, size);

calloc() 函数说明: 在内存动态存储区中分配 n 块长度为“size”字节的连续区域。函数返回值为该区域的首地址。calloc() 与 malloc() 的区别在于 calloc() 可分配 n 块区域。

例如:

```
struct student * p;
```

```
p = (struct student *) calloc(2, size(struct student));
```

其中 size(struct student) 是求 student 的结构长度, p 是一个指向结构 student 的结构指针变量, 按 student 的长度分配两块连续区域, 并把其首地址赋给 p。

3. 内存空间释放函数 free()

free() 函数的一般形式为:

```
free(void * p)
```

free() 说明: 释放 p 所指向的一块内存空间。p 是一个任意类型的指针变量, 它指向被释放区域的首地址。被释放的应是由 malloc() 和 calloc() 函数所分配的区域。

【例 10.8】 分配一块区域, 实现例 10.1 的功能。

【参考代码】

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
struct student
```

```
{
```

```
int id;
```

```
char name[20];
```

```
char sex;
```

```
int age;
```

```
float score[3];
```

```
};
```

```
void main()
```

```
{
```

```
struct student * p;
```

```

int i;
p = (struct student *) malloc(sizeof(struct student));
printf("请输入一学生的学号、姓名、性别、年龄:\n");
scanf("%d%s", &p->id, p->name);
getchar(); //接收输入时用于姓名和性别之间的回车符
p->sex = getchar();
scanf("%d", &p->age); //输入年龄
printf("请输入该学生三门课程成绩:\n");
for(i = 0; i < 3; i++)
    scanf("%f", &p->score[i]);
printf("输入的信息如下:\n");
printf("学号:%d\n 姓名:%s\n 性别:%c\n 年龄:%d\n",
p->id, p->name, p->sex, p->age);
printf("成绩:");
for(i = 0; i < 3; i++)
    printf("%5.1f", p->score[i]);
printf("\n");
free(p);
}

```

程序运行结果如图 10.5 所示。

【程序说明】

本程序先定义一个外部结构 student, 在主函数内定义了一个指向结构的指针变量, 用语句“p = (struct student *) malloc(sizeof(struct student));”分配一块 student 的内存空间, 并把首地址赋予 p, 然后输入一名学生的信息并输出, 最后释放 p 所指向的空间。

10.5.2 链表操作

使用指针可以创建复杂的数据结构, 如链表。链表是一种常见的、重要的数据结构, 它是动态分配存储的一种结构, 也是利用指针链在一起的线性结构。

1. 链表的定义

在例 10.8 中采用动态分配的办法为一个结构分配内存空间, 每一次分配一块空间可用来存放一名学生的数据, 在计算机专业词汇中称为一个“结点”。有多少名学生就应该申请分配多少块内存空间, 也就是说要建立多少个结点。当然也可用结构数组实现, 但无法动态分配或释放某名学生信息所占用的空间。而用动态存储的方法就能很好地解决这个问题。

数组在内存中是一块连续的区域, 而动态存储则不需要连续区域, 即每个结点可以不连续, 这样增加、删除一个结点非常方便。但要求每个结点中必须要有一个存放下一个结点的指针(或指针域), 用以把所有结点“链”成一串。单向链表示意图如图 10.7 所示。



图 10.7 单向链表示意图

图中的第 0 个结点称为头结点,存放第 1 个结点的首地址,它没有数据,只是一个指针变量。后面的每个结点都分两个域,一个是数据域,存放各种实际的数据,如学号、姓名、性别等。另一个为指针域,存放下一结点的首地址。“链”中的每一个结点都是同一个结构类型。例如,一个存放学生信息的结点为如下结构:

```
struct student
{
    int id;
    char name[20];
    char sex;
    int age;
    float score;
    struct student * next;
}
```

其中前 5 个成员组成数据域,最后一个成员 next 构成指针域,它是一个指向 student 类型结构的指针变量。

2. 链表的基本操作

对链表的主要操作有以下几种:

- (1) 建立链表;
- (2) 结点的查找与输出;
- (3) 插入一个结点;
- (4) 删除一个结点。

下面通过例题来说明这些操作。

【例 10.9】 建立一个三个结点的链表,存放学生数据。为简单起见,假定学生数据结构中只有学号、姓名和成绩三项,它们属于不同数据类型,具有一定代表性。可编写一个建立链表的函数 creat()。

【参考代码】

```
#include <stdio.h>
#include <stdlib.h>
#define NULL 0
#define TYPE struct student
#define LEN sizeof (struct student)
struct student
{
    int id;
    int name[20];
    float score;
    struct student * next;
};
```

```
TYPE * creat(int n)
```

```
{
    struct student * head, * pf, * pb;
```



图 10.10 单链表示意图

```

int i;
printf("请输入学号、姓名、成绩:\n");
for(i=0;i<n;i++)
{
    pb = (TYPE *) malloc(LEN); //分配结点空间
    scanf("%d%s",&pb->id,pb->name);
    getchar(); //可以没有这个语句
    scanf("%f",&pb->score);
    if(i==0)
        pf = head = pb;
    else
        pf->next = pb;
    pb->next = NULL;
    pf = pb;
}
return(head);

void print(TYPE * head)
{
    printf("学号、姓名、成绩:\n");
    while(head != NULL)
    {
        printf("%4d%8s%8.1f\n",head->id,head->name,head->score);
        head = head->next;
    }
}

void main()
{
    TYPE * head;
    int n;
    printf("请输入学生人数:");
    scanf("%d",&n);
    head = creat(n);
    print(head);
}

```

程序运行结果如图 10.8 所示。

【程序说明】

在函数外首先用宏定义对三个符号常量作了定义。这里用 TYPE 表示 struct student, 用 LEN 表示 sizeof(struct student) 的主要目的是为了在以下程序内减少书写并使阅读更加方便。结构 student 定义为外部类型, 程序中的各个函数均可使用该定义。

creat() 函数用于建立一个有 n 个结点的链表, 它是一个指针函数, 它返回的指针指向 student 结构。在 creat 函数内定义了三

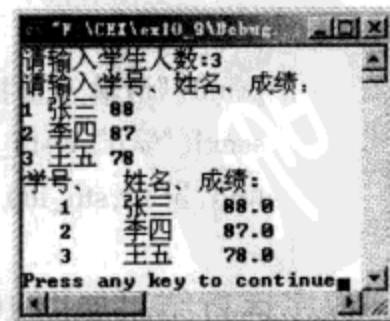


图 10.8 例 10.9 运行结果

个 student 结构的指针变量。head 为头指针, pf 为指向相邻结点的前一结点的指针变量, pb 为后一结点的指针变量。

【注意点】

(1) 语句“getchar();”可以不要,但如果后输入的字符型数据就该有此语句,用于接收字符串后的回车符。

(2) 预定义“#define NULL 0”可以用系统自己含有的 NULL。

(3) 预定义“#define TYPE struct student”和“#define LEN sizeof (struct student)”的使用请读者多思考。

(4) 函数 creat() 是返回值为指针的函数,返回值是一个地址(指针)。

(5) 函数 print() 实现学生信息的输出,语句“head = head -> next;”不能少。

【例 10.10】 编写一个函数在链表中按学号查找该结点,并输出该结点学生信息。

【参考代码】

```
..... //此处为例 10.9 中 creat() 和 print() 函数代码

void find(TYPE * head,int stu_id)
{
    TYPE * p;
    p = head;
    while(p -> id != stu_id && p -> next != NULL)
        p = p -> next;
    if(p -> id == stu_id)
    {
        printf("查找到的学生信息为:\n");
        printf("学号:%4d\n 姓名:%8s\n 成绩:%8.1f\n",p -> id,p -> name,p -> score);
    }
    if(p -> id != stu_id && p -> next == NULL)
        printf("没有找到学号为:%d 的学生信息!\n",stu_id);
}

void main()
{
    TYPE * head;
    int n,stu_id;
    printf("请输入学生人数:");
    scanf("%d",&n);
    head = creat(n);
    print(head);
    printf("请输入查找的学生学号:");
    scanf("%d",&stu_id);
    find(head,stu_id);
}
```

程序运行结果如图 10.9 所示。

【程序说明】

本函数中使用的 TYPE 与例 10.9 相同,函数 find() 有两个形参,head 是指向链表的头指

针变量,stu_id 为查找的学生学号,进入 while 语句,逐个检查结点的 id 成员是否等于 stu_id,如果不等于 stu_id 且指针域不等于 NULL,则指针到下一个结点,如果找到该结点,则返回结点指针,输出查找到的学生信息。

【注意点】

(1) 查找循环语句的判断条件“p -> id != stu_id && p -> next != NULL”,不能简单写成“p -> id != stu_id”,要有“p -> next != NULL”,否则还要继续查找。

(2) 本查找函数 find() 还可以如下编写:

```
void * find( TYPE * head, int stu_id)
{
    TYPE * p;
    p = head;
    while( p -> id != stu_id && p -> next != NULL)
        p = p -> next;
    if( p -> id == stu_id)
        return( p );           //返回查找到的学生信息首地址
    if( p -> id != stu_id && p -> next == NULL)
        printf("没有找到学号为:%d 的学生信息!\n", stu_id);
}
```

在主函数中实现输出查找到的学生信息。

【例 10.11】 编写一个函数删除链表指定结点。

【解题思路】

删除一个结点有以下两种情况。

(1) 被删除结点是第一个结点,这种情况只需要使 head 指向第二个结点即可,即 head = pb -> next。

(2) 被删除的结点不是第一个结点。这种情况使被删除结点的前一结点指向被删除结点的后一结点即可,即 pf -> next = pb -> next。

【参考代码】

```
..... //此处为例 10.9 中 creat() 和 print() 函数代码
void delete( TYPE * head, int stu_id)
```

```
{
    TYPE * pf, * pb;
    if( head == NULL)
        printf("该链表为空!");
    pb = head;
    while( pb -> id != stu_id && pb -> next != NULL)
```

/* 当不是要删除的结点,而且也不是最后一个结点时,继续循环 */

```
pf = pb;
```

```
pb = pb -> next;
```



图 10.9 例 10.10 运行结果

```

    if( pb -> id == stu_id)
    {
        if( pb == head)
        {
            /* 若找到被删除的结点,且为第一个结点,则使 head 指向第二个结点 */
            head = pb -> next;
        }
        else
        {
            pf -> next = pb -> next;
            free( pb );
            printf( "结点被删除!\n" );
        }
    }
    else
    {
        printf( "结点没有找到!\n" );
    }
}

void main( )
{
    TYPE * head;
    int n, stu_id;
    printf( "请输入学生人数:" );
    scanf( "%d", &n );
    head = creat( n );
    printf( "请输入删除结点的学生学号:" );
    scanf( "%d", &stu_id );
    fflush( stdin );
    delete( head, stu_id );
    printf( "删除后的链表中学生信息为:\n" );
    print( head );
}

```

程序运行的结果如图 10.10 所示。

【注意点】

本程序也可采用返回值是指针的函数。

【例 10.12】 编写一个函数,在链表中指定位置插入一个结点。

【解题思路】

在链表的指定位置插入结点,要求链表本身必须是一个已按某种规律排好序的。例如,在学生数据链表中,要求按学号顺序插入一个结点,设被插入结点的指针为 pi。可在 4 种不同情况下

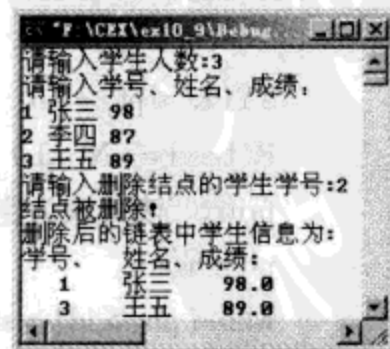


图 10.10 例 10.11 运行结果

插入。

(1) 原表为空表,只需要使 head 指向被插入结点即可。

(2) 被插入结点值最小,应插入第一结点之前。这种情况下使 head 指向被插入结点,被插入结点的指针域指向原来的第一结点即可。即:

```
pi->next = pb; head = pi;
```

(3) 在表尾插入。这种情况下,使原来末结点指针域指向被插入结点,被插入结点指针域置为 NULL。即:

```
pb->next = pi; pi->next = NULL;
```

(4) 在其他位置插入。这种情况下,使插入位置的前一结点的指针域指向被插入结点,使被插入结点的指针域指向插入位置的下一结点。即:

```
pi->next = pb; pf->next = pi;
```

【参考代码】

```
..... //此处为例 10.9 中 creat() 和 print() 函数代码
TYPE * insert( TYPE * head, TYPE * pi)
{
    TYPE * pf, * pb;
    pb = head;
    if( head == NULL)
    {
        head = pi;
        pi->next = NULL;
    }
    else
    {
        while( ( pi->id > pb->id ) && ( pb->next != NULL ) )
        {
            pf = pb;
            pb = pb->next;
        }
        //找插入位置

        if( pi->id <= pb->id )
        {
            if( head == pb )
            {
                head = pi;
                //在第一结点之前插入
            }
            else
            {
                pf->next = pi;
                pi->next = pb;
                //在其他位置插入
            }
        }
        else
        {
            pb->next = pi;
            //在表尾插入
        }
    }
}
```

```

    pi->next = NULL;
}

return head;

void main()
{
    TYPE * head, * pi;
    int n;
    printf("请输入学生人数:");
    scanf("%d", &n);
    head = creat(n);
    pi = (TYPE *) malloc(LEN);
    printf("请输入要插入的学生信息:\n");
    scanf("%d %s %f", &pi->id, pi->name, &pi->score);
    head = insert(head, pi);
    printf("插入结点后的链表中学生信息为:\n");
    print(head);
}

```

程序运行的结果如图 10.11 所示。

将上面的例 10.9 ~ 10.12 有机组织在一起, 建立一个完整的主函数调用各个子函数, 可实现链表的 4 种操作。其主函数 main() 如下:

```

void main()
{
    TYPE * head, * pi;
    int n;
    int stu_id;
    printf("请输入学生人数:");
    scanf("%d", &n);
    head = creat(n);
    print(head);
    printf("请输入查找的学生学号:");
    scanf("%d", &stu_id);
    find(head, stu_id);
    print(head);
    printf("请输入删除结点的学生学号:");
    scanf("%d", &stu_id);
    fflush(stdin); //清除键盘缓冲区
    deletel(head, stu_id);
    printf("删除后的链表中学生信息为:\n");
}

```

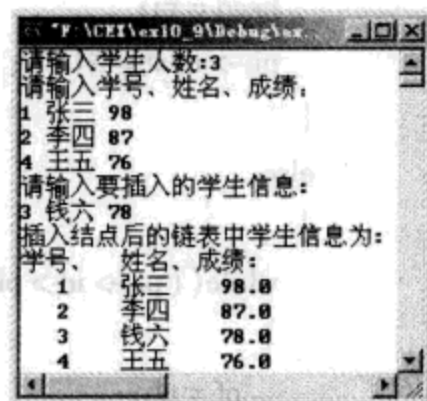


图 10.11 例 10.12 运行结果

```

print(head);
pi = (TYPE *) malloc(LEN);
printf("请输入要插入的学生信息:\n");
scanf("%d %s %f", &pi->id, pi->name, &pi->score);
head = insert(head, pi);
printf("插入结点后的链表中学生信息为:\n");
print(head);

```

10.6 共用

在 C 语言中,同一区域可供不同类型的数据使用,这些不同的数据,形成一个新的数据类型,称为共用。与结构类似,在共用内可以定义多种不同数据类型的成员,区别是共用类型变量所有成员共用一块内存单元(虽然每个成员都可以被赋值,但前面赋予的成员值被后面赋予的成员值所覆盖,只有最后一次赋予的成员值能够保存且有意义。

1. 共用的定义

共用内存空间,其定义一般形式为:

union 共用类型名

```

{
    成员序列;
    变量序列;
}

```

2. 共用变量的使用

例如,下面定义一个共用和一个结构,比较它们之间的区别。

union data1

```

{
    short i;
    int j;

```

double d;

char ch;

} a;

struct data2

```

{
    short i;
    int j;
    double d;
    char ch;
} b;

```

共用与结构的区别如图 10.12 所示。

【例 10.13】 区别共用与结构占用内存空间的大小。

#include <stdio.h>

union data1

【注意】

(1) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(2) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(3) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(4) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(5) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(6) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(7) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(8) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(9) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(10) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(11) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(12) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(13) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(14) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(15) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(16) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(17) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(18) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(19) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

(20) 共用占用的内存空间长度是其所含成员占用的内存空间长度。

```

{
    double d;
    short i;
    char ch;
} a;
struct data2
{
    double d;
    short i;
    char ch;
} b;
void main()
{
    printf("data1 占用字节数:%d\n", sizeof(a));
    printf("data2 占用字节数:%d\n", sizeof(b));
    printf("short 型数据占用字节数:%d\n", sizeof(short));
    printf("char 型数据占用字节数:%d\n", sizeof(char));
    printf("int 型数据占用字节数:%d\n", sizeof(int));
    printf("long 型数据占用字节数:%d\n", sizeof(long));
    printf("float 型数据占用字节数:%d\n", sizeof(float));
    printf("double 型数据占用字节数:%d\n", sizeof(double));
}

```



图 10.12 共用与结构的区别

程序运行结果如图 10.13 所示。

【注意点】

(1) 共用占用的内存空间长度是所有成员中占用内存空间长度最大者。

(2) 结构占用的内存空间长度是大于或等于 (Visual C++ 6.0 中是大于或等于, TC 中是等于) 各成员占用内存空间之和。

【例 10.14】 已知字符 '0' 的 ASCII 码值是 48, 且数组的第 0 个元素在低位, 观察程序的运行结果。

```

#include <stdio.h>
void main()
{
    union
    {
        int i[2];
        long k;
        char c[4];
    } r, *s = &r;

    s->i[0] = 0x39;
    s->i[1] = 0x38;
}

```



图 10.13 例 10.13 运行结果


```

printf("变量 r 占用内存空间长度: %d\n", sizeof(r));
printf("r.i[%d] 的地址: %p, r.i[%d] = %x\n", 0, &(r.i[0]), 0, r.i[0]);
printf("r.i[%d] 的地址: %p, r.i[%d] = %x\n", 1, &(r.i[1]), 1, r.i[1]);
printf("s->i[%d] 的地址: %p, s->i[%d] = %x\n", 0, &(s->i[0]), 0, s->i[0]);
printf("r.k 的地址: %p, r.k = %x\n", &(r.k), r.k);
printf("r.c[%d] 的地址: %p, r.c[%d] = %x\n", 0, &(r.c[0]), 0, r.c[0]);
printf("r.c[%d] 的地址: %p, r.c[%d] = %x\n", 1, &(r.c[1]), 1, r.c[1]);
printf("r.c[%d] 的地址: %p, r.c[%d] = %x\n", 2, &(r.c[2]), 2, r.c[2]);
printf("r.c[%d] 的地址: %p, r.c[%d] = %x\n", 3, &(r.c[3]), 3, r.c[3]);
printf("r.c[%d] 的地址: %p, r.c[%d] = %x\n", 4, &(r.c[4]), 4, r.c[4]);
printf("r.c[%d] 的地址: %p, r.c[%d] = %x\n", 5, &(r.c[5]), 5, r.c[5]);
printf("r.c[%d] 的地址: %p, r.c[%d] = %x\n", 6, &(r.c[6]), 6, r.c[6]);
printf("r.c[%d] 的地址: %p, r.c[%d] = %x\n", 7, &(r.c[7]), 7, r.c[7]);

```

程序运行结果如图 10.14 所示。

【注意点】

(1) `r.i[0]` 和 `r.i[1]` 这两个数组元素占用内存空间 8 字节, 也就是说共用的成员是数组的元素占用连续的内存空间。

(2) 每个成员的首地址是相同的。

(3) 数组的下标可以超过数组元素个数, 如 `r.i[3]`、`r.i[4]` 也不算错, 但不知是何值。

(4) 其成员空间分配情况如图 10.15 所示。



图 10.14 例 10.14 运行结果

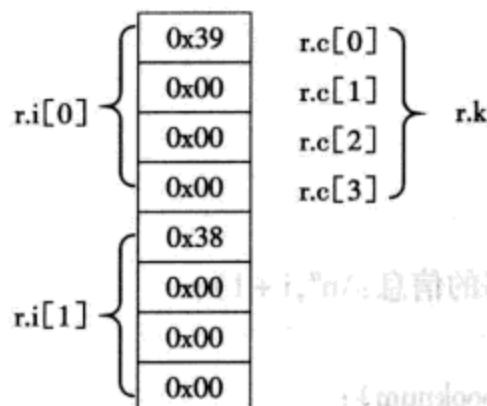


图 10.15 例 10.14 内存分配示意图

3. 使用共用变量

对共用变量的赋值和使用都是对变量的成员进行的, 其一般形式为:

共用变量名. 成员名

注意, 共用变量的赋值不能使用如下形式:

共用变量名 -> 成员名

否则, 会产生编译错误。例如, 将例 10.14 中的“`r.i[0]`”改为“`r->i[0]`”, 则会产生“error C2232: ‘->i’: left operand has ‘union’ type, use .”编译错误。

使用共用类型数据时, 应注意以下特点。

(1) 同一内在段可以用来存放不同类型的成员,但是每一瞬时只能存放其中的一种(只有一种有意义)。

(2) 共用变量中有意义的成员是最后一次存放的成员。例如,在“a.i=1;a.ch='A';a.d=3.14;”语句后,只有“a.d=3.14;”有意义(a.i、a.ch 也可以访问,但无实际意义)。

(3) 共用变量的地址和其成员的地址都是同一地址,即 &a.i = &a.ch = &a.d = &a。

(4) 除整体赋值外,不能对共用变量进行赋值,也不能企图通过引用共用变量来得到成员的值。不能在定义共用变量时对共用变量进行初始化(系统不清楚是为哪个成员赋初值)。

(5) 可以将共用变量作为函数参数,函数也可以返回共用,共用指针。

(6) 共用和结构可以互相嵌套。

【例 10.15】 利用结构知识编写“图书管理系统”的“图书增加”模块。

【参考代码】

```
#include <stdio.h>
```

```
#define N 2
```

```
struct book
```

```
{
```

```
int booknum;
```

```
char bookname[20];
```

```
char bookauthor[20];
```

```
char press[50];
```

```
float price;
```

```
int count;
```

```
};
```

```
void bookadd()
```

```
{
```

```
struct book book1[N];
```

```
int i;
```

```
for(i=0;i<N;i++)
```

```
{
```

```
printf("请输入第%d本书的信息:\n",i+1);
```

```
printf("书号:");
```

```
scanf("%d",&book1[i].booknum);
```

```
fflush(stdin);
```

```
printf("书名:");
```

```
gets(book1[i].bookname);
```

```
fflush(stdin);
```

```
printf("作者:");
```

```
gets(book1[i].bookauthor);
```

```
fflush(stdin);
```

```
printf("出版社:");
```

```
gets(book1[i].press);
```

```
fflush(stdin);
```

```
printf("书价:");
```

//定义书的类型

//书号

//书名

//作者

//出版社

//书价

//剩余本数

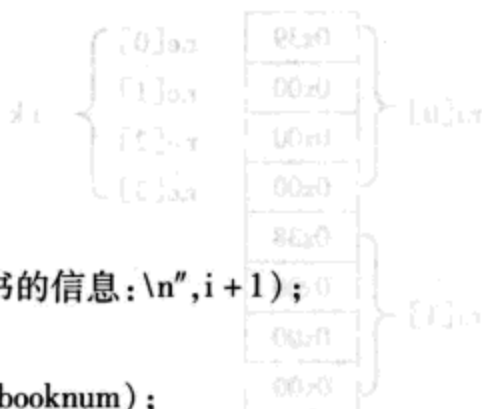


图 10.15 图书增加模块的内存布局



```

scanf("%f",&book1[i].price);
printf("剩余本数:");
scanf("%d",&book1[i].count);
}
printf("输出录入的图书信息:\n");
printf("书号 书名 作者 出版社 书价 剩余本数\n");
for(i=0;i<N;i++)
{
    printf("%6d",book1[i].booknum);
    printf("%s",book1[i].bookname);
    printf("%s",book1[i].bookauthor);
    printf("%s",book1[i].press);
    printf("%6.1f",book1[i].price);
    printf("%6d\n",book1[i].count);
}
}
void main()
{
    bookadd(); //调用增加图书模块
}

```

【注意点】

(1) 可以用 typedef 定义结构:

```

#define N 3
typedef struct book
{
    ...//代码省略
} BOOK;
...
void main()
{
    BOOK book1[N];
}

```

(2) 也可以用预定义来定义结构:

```

#define TYPE struct book
struct book
{
    ...//代码省略
}
void main()
{ TYPE book1[N]; }

```

10.7 位运算

在计算机内部,数据的存储、运算都是以二进制进行的。位运算是指进行二进制位的

运算。

位运算的操作对象一般是整型或字符型。位运算是 C 语言的低级语言特性,广泛应用于对底层硬件、外围设备的状态检测和控制。

1. 位运算符

C 语言提供如表 10.1 所示的位运算符。

表 10.1 位运算符

运算符	含义	运算符	含义
&	位与	~	取反
	位或	<<	左移
^	异或	>>	右移

说明:

(1) 位运算符中除“~”外,均为二目(元)运算符,即要求两侧各有一个数据。

(2) 数据只能是整型或字符型的,不能是实型数据。

2. 位运算

1) 位与(&)

该运算符表示将其两边的数据对应的二进制位按位进行“与”运算。二者全为 1,结果为 1,否则为 0。

例如:

$x = 10111101 (0xBD)$

$y = 01101000 (0x68)$

则 $x \& y = 00101000 (0x28)$

2) 位或(|)

该运算符表示将其两边的数据对应的二进制位按位进行“或”运算。二者只要有一个为 1,结果为 1;否则,结果为 0。

例如:

$x = 10111101 (0xBD)$

$y = 01101000 (0x68)$

则 $x | y = 11111101 (0xFD)$

3) 取反(~)

该运算符表示对一个二进制数的每一位都取反。

例如:

$x = 10111101 (0xBD)$

$\sim x = 01000010 (0x42)$

4) 左移(<<)

左移(<<)的功能是将一个数的各个二进制位全部向左平移若干位(左边移出的部分忽略,右边补 0)。每左移 1 位,相当于乘以 2,左移 n 位相当于乘以 2^n 。

例如:二进制数 10111101 左移 1 位,结果为 101111010。

```
unsigned char a = 25; /* (25)10 = (0001,1001)2 */
a << 2; /* (01100100)2 = (100)10 */
```

5) 右移(>>)

右移(>>)的功能是将一个数的各个二进制位全部向右平移若干位(右边移出的部分忽略,左边对无符号数补0,有符号数补符号位)。每右移1位,相当于除以2,右移n位相当于除以 2^n 。

例如:

```
unsigned char a = 0x7B; /* (123)10 = (0111,1011)2 = (7B)16 */
a >> 2; /* (00011110)2 = (30)10 */
```

6) 异或(^)

异或(^)的功能:相应位,相同为0,相异为1。

例如:

```
x = 10111101 (0xBD)
y = 01101000 (0x68)
```

则 $x \oplus y = 11010101 (0xD5)$

【例 10.16】 将一个十进制数转化为二进制数。

【解题思路】

C语言中没有二进制输出格式,人工转换方法为设置一个屏蔽位,其中只有1个1,其余为0,为1的位为测试位置。将此屏蔽字与被转换数进行“位与”运算,根据运算结果判断被测试的位是1还是0,循环测试(VC中一个整数4个字节,32位,测试32次,从最高位开始测试,每次测试后屏蔽字右移1位,以便测试下一位),输出的测试结果就是十进制整数对应的二进制数。

【参考代码】

```
#include <stdio.h>

void main()
{
    int i, bit; /* i 为循环变量, bit 为位 1/0 标志变量 */
    unsigned int n, mask; /* n 为要转换的整数, mask 为屏蔽字变量 */
    mask = 0x80000000; /* 初始屏蔽字 1000,0000,0000,0000,0000,0000,0000,0000 */
    printf("请输入一个整数:");
    scanf("%d", &n);
    printf("%d 的二进制表示为:\n", n);
    for(i = 0; i < 32; i++)
    {
        if(i % 4 == 0 && i != 0)
            printf(","); /* 二进制每 4 位用“,”分隔,便于查看 */
        bit = (n & mask) ? 1 : 0;
        printf("%1d", bit); /* 输出 1 或 0 */
        mask = mask >> 1; /* 右移 1 位 */
    }
    printf("\n");
}
```

程序运行结果如图 10.16 所示。

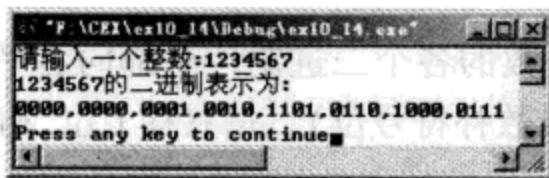


图 10.16 例 10.16 运行结果

本章小结

结构、共用的定义和使用方法,类型和变量的区别,链表都是本章的重点,有关链表的操作是本章的难点。读者除了掌握结构、共用的基本概念和使用方法外,还需要掌握类型定义方法和动态链表的建立方法。

(1) 主要介绍了结构类型的定义,结构变量的定义,结构成员占用内存的情况,结构变量的引用与初始化,结构和数据之间的关系,结构与指针之间的关系,结构与函数之间的关系,用结构来处理链表。

(2) 单向链表的操作较难,书中给出了每个操作的实例,读者可以上机改造书中例子,并体会其 4 种链表操作的精髓。

(3) 熟悉共用与结构之间的区别、共用类型的定义、共用变量的定义、共用成员占用内存情况、共用变量的引用和初始化。

(4) 简单介绍了位运算操作。

习题十

一、选择题

1. 设有以下语句

```
typedef struct TT
{ char c; int a[4]; } CIN;
```

则下面叙述中正确的是()。

- A) 可以用 TT 定义结构体变量
- C) 可以用 CIN 定义结构体变量

- B) TT 是 struct 类型的变量
- D) CIN 是 struct TT 类型的变量

2. 根据以下的定义,能输出字母 M 的语句是()。

```
struct person
{ char name[9];
  int age; }
struct person class[10] = { "John", 17,
                             "Paul", 19,
                             "Mary", 18,
```


"Adam", 16, } ;

- A) printf("%c\n", class[3].name); B) printf("%c\n", class[3].name[1]);
C) printf("%c\n", class[2].name[1]); D) printf("%c\n", class[2].name[0]);

3. 有以下结构体说明、变量定义和赋值语句

```
struct STD
```

```
{ char name[10];
```

```
  int age;
```

```
  char sex;
```

```
  s[5], *ps;
```

```
  ps = &s[0];
```

则以下 scanf() 函数调用语句中错误引用结构体变量成员的是()。

- A) scanf("%s", s[0].name); B) scanf("%d", &s[0].age);
C) scanf("%c", &(ps -> sex)); D) scanf("%d", ps -> age);

4. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
  struct cmplx
```

```
  {
```

```
    int x;
```

```
    int y;
```

```
  } cnum[2] = {1, 3, 2, 7};
```

```
  printf("%d\n", cnum[0].y/cnum[0].x * cnum[1].x);
```

```
}
```

- A) 0 B) 1 C) 3 D) 6

5. 若有以下定义和语句:

```
union data
```

```
{ int i; char c; float f; } x;
```

```
int y;
```

则以下语句正确的是()。

- A) x = 10.5; B) x.c = 101; C) y = x; D) printf("%d\n", x);

6. 若有以下说明和语句, 则值为 6 的表达式是()。

```
struct st
```

```
{ int n;
```

```
  struct st *next;
```

```
};
```

```
struct st a[3], *p;
```

```
a[0].n = 5; a[0].next = &a[1];
```

```
a[1].n = 7; a[1].next = &a[2];
```

```
a[2].n = 9; a[0].next = '\0';
```

```
p = &a[0];
```

A) $P++ \rightarrow n$ B) $p \rightarrow n++$ C) $(*p).n++$ D) $++p \rightarrow n$

7. 有以下程序:

```
#include <stdio.h>
```

```
#include <string.h>
```

```
typedef struct
```

```
{
```

```
    char name[9];
```

```
    char sex;
```

```
    float score[2];
```

```
} STU;
```

```
void f(STU a)
```

```
{
```

```
    STU b = {"Zhao", 'm', 85.0, 90.0};
```

```
    int i;
```

```
    strcpy(a.name, b.name);
```

```
    a.sex = b.sex;
```

```
    for(i = 0; i < 2; i++)
```

```
        a.score[i] = b.score[i];
```

```
}
```

```
void main()
```

```
{
```

```
    STU c = {"Qian", 'f', 95.0, 92.0};
```

```
    f(c);
```

```
    printf("%s, %c, %2.0f, %2.0f\n", c.name, c.sex, c.score[0], c.score[1]);
```

```
}
```

程序的运行结果是()。

A) Qian, f, 95, 92 B) Qian, m, 85, 90 C) Zhao, f, 95, 92 D) Zhao, m, 85, 90

8. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
typedef union { long x[2];
```

```
    int y[4];      //16 字节
```

```
    char z[8];
```

```
} MYTYPE;
```

```
MYTYPE them;
```

```
void main()
```

```
{ printf("%d\n", sizeof(them)); }
```

A) 32

B) 16

C) 8

D) 24

9. 设有以下定义:

```
union data
{ int d1; float d2; } demo;
```

则下面叙述中错误的是()。

- A) 变量 demo 与成员 d2 所占的空间的内存字节数相同
- B) 变量 demo 中各成员的地址相同
- C) 变量 demo 和各成员的地址相同
- D) 若给 demo.d1 赋 99 后, demo.d2 中的值是 99.0

10. 以下程序的输出结果是()。

```
#include <stdio.h>

struct st
{ int x;
  int *y;
  *p;
} dt[4] = {10, 20, 30, 40};

struct st aa[4] = {50, &dt[0], 60, &dt[0], 60, &dt[0], 60, &dt[0]};

void main()
{ p = aa;
  printf("%d\n", ++p->x);
  printf("%d\n", ( ++p) ->x);
  printf("%d\n", ++(*p->y));
}
```

- | | | | |
|-------|-------|-------|-------|
| A) 10 | B) 50 | C) 51 | D) 60 |
| 20 | 60 | 60 | 70 |
| 20 | 21 | 11 | 31 |

11. 若已建立下面的链表结构, 指针 p, s 分别指向图 10.17 中所示结点, 则不能将 s 所指的结点插入到链表末尾的语句是()。

- A) s->next = NULL; p = p->next; p->next = s;
- B) p = p->next; s->next = p->next; p->next = s;
- C) p = p->next; s->next = p; p->next = s;
- D) p = (*p).next; (*s).next = (*p).next = s;

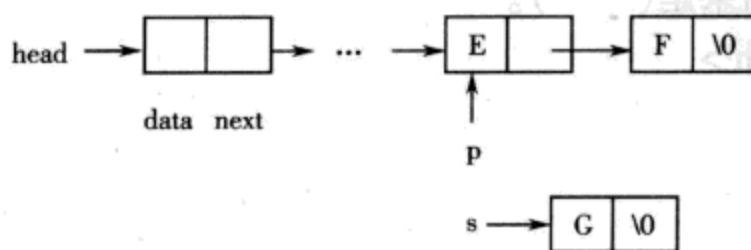


图 10.17

12. 有以下程序:

```
#include <stdio.h>
#include <stdlib.h>
int fun(int n)
{ int *p;
  p = (int *) malloc(sizeof(int));
  *p = n;
  return *p;
}
void main()
{ int a;
  a = fun(10);
  printf("%d\n", a + fun(10)); }
```

程序的运行结果是()。

A) 0

B) 10

C) 20

D) 出错

13. 有以下程序:

```
#include <stdio.h>
struct st
{ int x, y; } data[2] = { 1, 10, 2, 20 };
void main()
{
  struct st *p = data;
  printf("%d, ", p->y); printf("%d\n", (++p)->x);
}
```

程序的运行结果是()。

A) 10, 1

B) 20, 1

C) 10, 2

D) 20, 2

14. 以下程序的输出结果是()。

```
#include <stdio.h>
void main()
{ char x = 040;
  printf("%d\n", x = x << 1);
}
```

A) 100

B) 160

C) 120

D) 64

15. 以下程序的输出结果是()。

```
#include <stdio.h>
void main()
{
  int a = 15, b, c;
  b = a << 2;
  c = a * 4;
  printf("%d, %d, %d\n", a, b, c);
}
```

A) 15, 15, 15 B) 15, 60, 60 C) 60, 60, 60 D) 15, 60, 15

16. 以下程序的输出结果是()。

```
#include <stdio.h>
```

```
void main()
```

```
{ int x=35; char z='A';
```

```
printf("%d\n", (x&15)&&(z<'a'));
```

A) 0

B) 1

C) 2

D) 3

17. 假定已建立以下链表结构,且指针 p 和 q 已指向如图 10.18 所示的结点。

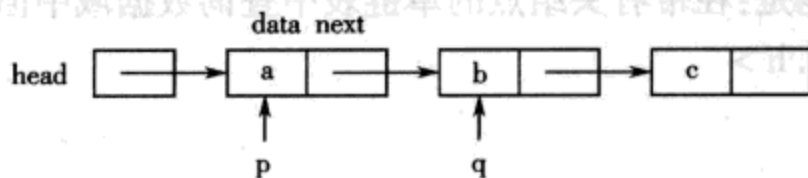


图 10.18 指针 p 和 q 指向图

则以下选项中可将 q 所指结点从链表中删除并释放该结点的语句组是()。

A) (*p).next = (*q).next; free(p);

B) p = p->next; free(q);

C) p = q; free(q);

D) p->next = q->next; free(q);

二、填空题

1. 为建立如图 10.17 结构(每个节点有二个域:data 是数据域,next 是指针域)请填空。

```
struct link
```

```
{ char data;
```

```
node;
```

2. 以下程序中的函数 fun() 的功能是:统计 person 所指结构体数组中所有性别(sex)为 M 的记录个数,存入变量 n 中,并作为函数的返回值,请填空。

```
#include <stdio.h>
```

```
#define N 3
```

```
typedef struct
```

```
int num;
```

```
char nam[10];
```

```
char sex;
```

```
} SS;
```

```
int fun(SS person[])
```

```
{
```

```
int i, n=0;
```

```
for(i=0; i<N; i++)
```

```
if(_____=='M')
```

```
n++;
```



```
int min (struct node * first)
```

```
    struct node * p;
```

```
    int m;
```

```
    p = first -> next;
```

```
    m = p -> data;
```

```
    for (p = p -> next; p != '\0'; p = _____)
```

```
        if( _____ ) m = p -> data;
```

```
    return m;
```

```
}
```

5. 以下函数 creat() 用来建立一个带有头结点的单向链表, 新产生的结点总是插在链表的末尾, 单向链表的头指针作为函数值返回。请填空。

```
#include <stdio.h>
```

```
struct list
```

```
{ char data;
```

```
  struct list * next;
```

```
};
```

```
struct list * creat()
```

```
{ struct list * h, * p, * q;
```

```
  char ch;
```

```
  h = _____ malloc( sizeof( _____ ) );
```

```
  p = q = h;
```

```
  ch = getchar();
```

```
  while( ch != '?' )
```

```
  { p = _____ malloc( sizeof( _____ ) );
```

```
    p -> data = ch;
```

```
    q -> next = p;
```

```
    q = p;
```

```
    ch = getchar();
```

```
  }
```

```
  p -> next = '\0';
```

```
  _____;
```

```
}
```

6. 以下程序中 c 的二进制值是_____。

```
char a = 3, b = 6, c;
```

```
c = a^b << 2;
```

7. 运用位运算, 能将变量 ch 中的大写字母转换成小写字母的表达式是_____。

三、编程题

1. 有 3 名学生, 每名学生信息包括姓名、学号、成绩。要求找出成绩最高的学生的姓名、学

号和成绩。

2. 有两名学生,每名学生的数据包括姓名、年龄、性别和学号。要求在 main() 函数中输入这两名学生的数据,然后输出。

3. 已知 head 指向一个带有头结点的单向链表,链表中每个结点包含数据域(data)和指针域(next),数据域为整型。请分别编写函数,在链表中查找数据域值最大的结点。

(1) 由函数值返回找到的最大值。

(2) 由函数值返回最大值所在结点的地址值。

2. 以下函数 creat() 用来建立一个带有头结点的单向链表,链表中每个结点包含数据域(data)和指针域(next),数据域为整型。请编写函数,在链表中查找数据域值最大的结点。

函数 creat() 的函数原型如下:

```
#include <stdio.h>

struct list
{
    char data;
    struct list * next;
};

struct list * creat();
```

```
struct list * h, * p, * q;
char ch;
h = (struct list *) malloc(sizeof(struct list));
p = q = h;
do
{
    ch = getch();
    while(ch != '\n')
    {
        p = (struct list *) malloc(sizeof(struct list));
        p->data = ch;
        p->next = q;
        q = p;
    }
    ch = getch();
} while(ch != '\n');
```

6. 以下函数 creat() 用来建立一个带有头结点的单向链表,链表中每个结点包含数据域(data)和指针域(next),数据域为整型。请编写函数,在链表中查找数据域值最大的结点。

```
int a=3, b=6, c;
c = a << 2;
```

7. 运用位运算,能将变量 ch 中的大写字母转换成小写字母,请编写函数实现。

三、编程题

1. 有 3 名学生,每名学生的信息包括姓名、学号、成绩。要求编写函数,在链表中查找成绩最高的学生的信息。

第 11 章 文件

前面各章进行数据处理时,无论数据量有多大,每次运行程序都必须通过键盘输入,程序处理的结果也只能输出到屏幕上。如果将输入或输出的数据以磁盘文件的形式存储起来,则在大批量数据处理时将会十分方便。为此,需要学习文件有关的概念、文件指针和文件操作等相关知识。

11.1 文件概述

11.1.1 文件及分类

文件是计算机中一个重要概念,通常是指存储在外部介质上的数据的集合。这个数据集有一个名称,叫做文件名。文件名的命名规则与 C 标识符相同。前面各章中已经多次使用了文件,例如源程序文件、目标文件、可执行文件、库文件(头文件)等。

从不同的角度可对文件作不同的分类。

(1) 按文件所依附的介质分类,有卡片文件、纸带文件、磁带文件和磁盘文件等。

(2) 按文件内容来分类,有源文件、目标文件和数据文件等。

(3) 按文件的数据组织形式分类,有字符文件和二进制文件。

(4) 按文件操作可划分为输入文件、输出文件和输入/输出文件。

字符文件又称 ASCII 文件,也称为文本文件,这种文件在磁盘中存放时每个字符对应一个字节,用于存放对应的 ASCII 码。

例如,整数 1234 的存储形式如下,共占用 4 个字节。

ASCII 码: 00000001 00000010 00000011 00000100

↓ ↓ ↓ ↓

十进制码: 1 2 3 4

ASCII 码文件可在屏幕上按字符显示。例如,源程序文件就是 ASCII 文件,用记事本或 DOS 命令 TYPE 可显示文件的内容。由于是按字符显示,因此能读懂文件内容。

二进制文件是按二进制的编码方式来存放文件的。

例如,存储整数 1234 时,先将十进制 1234 转化为二进制 10011010010 后再存储。其存储形式为:

00000000 00000000 00000100 11010010

只占 4 个字节(Visual C++ 6.0)。二进制文件虽然也可在屏幕上显示,但其内容无法读懂。

输入/输出字符流的开始和结束只由程序控制而不受物理符号(如回车符)的控制,因此也把这种文件称作“流式文件”。

文本文件、二进制文件不是用后缀来确定的,而是以内容来确定的,但是文件后缀往往隐

含其类别,如 *.txt 代表文本文件,*.exe 代表二进制文件等。本章讨论流式文件的打开、关闭、读、写、定位等各种操作。

11.1.2 文件类型指针

1. FILE 类型

FILE 类型是一种结构类型,这个结构中包含缓冲区(内存的区域)的大小、文件的状态标志、文件的描述符等信息。该结构类型在 C 语言系统中率先定义,包含在头文件“stdio.h”中。FILE 结构类型如下:

```
typedef struct
{
    short    level;        /* 缓冲区“满”或“空”的标志 */
    unsigned flags;        /* 文件状态标志 */
    char     fd;           /* 文件描述符 */
    unsigned char hold;     /* 若无缓冲区不读取字符 */
    short    basize;        /* 缓冲区大小 */
    unsigned char *buffer;  /* 数据缓冲区位置 */
    unsigned char *curp;    /* 当前活动指针 */
    unsigned istemp;        /* 临时文件描述符 */
    short    token;        /* 用于有效性检查 */
} FILE;
```

程序每使用一个文件,系统就为此文件设置一个 FILE 类型变量。也就是说,程序使用几个文件,系统就设置几个 FILE 类型变量,存放各个文件的相关信息。

2. 文件指针

在 C 语言中用一个指针变量指向一个文件,这个指针就称为文件指针。通过文件指针可以对它所指的文件进行各种操作。

定义文件指针的一般形式为:

FILE * 指针变量标识符;

其中,FILE 应为大写。

事实上,只需要使用文件指针完成文件的操作,不需要关心文件类型变量的内容。在打开一个文件后,系统开辟一个缓冲区,用来存放文件的属性状态(如文件的名称、文件的性质和文件当前状态等信息),这些信息利用一个结构类型指针变量存放,这个指针变量称为文件的指针变量,所有对文件的操作都通过此文件指针变量完成,直到文件关闭,文件指针指向的文件类型变量释放。例如:

FILE *fp, *fq;

如 fp 是表示指向 FILE 结构的指针变量,通过 fp 即可找到存放某个文件信息的结构变量,然后按结构变量提供的信息找到该文件,实施对文件的操作。习惯上把 fp 称为指向一个文件的指针。

11.1.3 文件的打开与关闭

文件在进行读写操作之前要先打开,使用完毕要关闭。

1. 文件的打开(fopen()函数)

文件打开后才能进行操作。文件打开通过调用 fopen() 函数实现。

fopen() 函数用来打开一个文件。其一般形式为:

文件指针名 = fopen(文件名, 使用文件方式);

其中:

(1) “文件指针名”必须是被定义为 FILE 类型的指针变量;

(2) “文件名”是被打开文件的文件名, 并由字符串常量或字符串数组构成;

(3) “使用文件方式”是指文件的类型和操作要求。

例如:

```
FILE *fp;
```

```
fp = fopen("d:\a.txt", "r");
```

说明:

(1) 打开文件 d: 盘根目录下 a.txt 文件, 只允许进行“读”操作。

(2) fopen() 函数返回指向 d:\a.txt 的文件指针, 并赋给 fp, fp 指向该文件。

(3) 关于文件要注意: 文件名包含文件名和扩展名, 路径要用“\”表示。

(4) 使用文件的方式有 12 种。文件打开方式见表 11.1 所示。

表 11.1 文件的打开方式

文件打开方式	含 义
"r"	只读打开一个文本文件, 只允许读数据
"w"	只写打开或建立一个文本文件, 只允许写数据
"a"	追加打开一个文本文件, 并在文件末尾写数据
"rb"	只读打开一个二进制文件, 只允许读数据
"wb"	只写打开或建立一个二进制文件, 只允许写数据
"ab"	追加打开一个二进制文件, 并在文件末尾写数据
"r+"	读写打开一个文本文件, 允许读和写
"w+"	读写打开或建立一个文本文件, 允许读和写
"a+"	读写打开一个文本文件, 允许读, 或在文件末追加数据
"rb+"	读写打开一个二进制文件, 允许读和写
"wb+"	读写打开或建立一个二进制文件, 允许读和写
"ab+"	读写打开一个二进制文件, 允许读, 或在文件末追加数据

对于文件使用方式有以下几点说明。

(1) 文件使用方式由“r”、“w”、“a”、“t”、“b”、“+”六个字符拼成, 各字符的含义如下:

r(read): 读

w(write): 写

a(append): 追加

t(text): 文本文件, 可省略不写

b(binary): 二进制文件

+: 读和写

(2) 凡用“r”打开一个文件时,该文件必须已经存在,且只能从该文件读出。

(3) 用“w”打开的文件只能向该文件写入。若打开的文件不存在,则以指定的文件名建立该文件;若打开的文件已经存在,则将该文件删去,重建一个新文件。

(4) 若要向一个已存在的文件追加新的信息,只能用“a”方式打开文件,此时该文件存在则追加,文件不存在则新建。

(5) 在打开一个文件时,如果出错,fopen()将返回一个空指针值 NULL。在程序中可以用这一信息来判别是否完成打开文件的工作,并作相应的处理。因此常用以下程序段打开文件:

```
if( (fp = fopen("d:\\a1.txt", "rt") == NULL)
{
    //“\\”第一个“\”是转义,第二个“\”表示根目录
    printf("\n 打开文件 d: \\ a1.txt 错误!");
    getch(); //从键盘输入一个字符,但不在屏幕上显示
    exit(1);
}
```

2. 文件关闭函数(fclose()函数)

文件一旦使用完毕,应用关闭文件函数把文件关闭,以避免文件的数据丢失等错误。

fclose()函数调用的一般形式如下:

fclose(文件指针);

例如:

fclose(fp);

正常完成关闭文件操作时,fclose()函数返回值为0。如返回非零值,则表示有错误发生。

11.2 文本文件的读写

11.2.1 字符输入/输出函数 fgetc()和 fputc()及文件结束检测函数 feof()

字符读写函数是以字符(字节)为单位的读写函数。每次可从文件读出或向文件写入一个字符。

1. 读字符函数 fgetc()

fgetc()函数的功能是从指定的文件中读一个字符。函数的一般形式为:

字符变量 = fgetc(文件指针);

其功能是从 fp 所指向的文件读一个字符,字符由函数返回。返回的字符赋给字符变量,也可以直接参与表达式运算。输入成功,返回输入字符,遇到文件结束,返回 EOF(-1)。

说明:

(1) 在 fgetc()函数调用中,读取的文件必须是以读或读写方式打开的。

(2) 每次读入一个字符,文件位置指针自动指向下一字节。

(3) 文本文件内全部是 ASCII 字符,其值不可能是 EOF(-1),所以可以使用 EOF(-1)确定文件结束。但是对于二进制文件不能这样做,因为可能在文件中间某个字符的值恰好等于-1,如果此时使用-1判断文件结束是不恰当的。为了解决这个问题,ANSI C 提供了 feof

(fp) 函数判断文件是否真正结束。

【例 11.1】 将 f: 盘上的 aa.txt 文件内容读出并在屏幕上显示。

【参考代码】

```
#include <stdio.h>
#include <stdlib.h> //exit(1)所在的头文件
void main()
{
    FILE *fp;
    char file1[20];
    printf("请输入文本文件名及位置:");
    scanf("%s", file1);
    if((fp = fopen(file1, "r")) == NULL) //判断文件能否打开
    {
        printf("不能打开文件: %s", file1);
        exit(1);
    }
    printf("文件%s 中的信息为:\n", file1);
    while(!feof(fp)) //若文件未结束
    {
        printf("%c", fgetc(fp)); //输出字符到屏幕上
    }
    printf("\n");
    fclose(fp); //关闭文件
}
```

程序运行结果如图 11.1 所示。

【注意点】

- (1) 输入的名可以带路径, 否则为当前默认路径。
- (2) 输入的文件可以为“f:\aa.txt”或“f: \aa.txt”。

2. 写字符函数 fputc()

fputc() 函数的功能是把一个字符写入指定的文件中。

函数调用的形式为:

fputc(字符, 文件指针);

其功能是将“字符”写入 fp 所指向的文件, 即向指定文件写入一个字符。如果输出成功, 函数返回值就是输出的字符; 如果不成功, 则返回一个 EOF(-1)。

说明: 每写入一个字符, 文件位置指针自动指向下一个字节。

【例 11.2】 读入文件 f:\aa.txt 的内容, 并复制到另一个文件中。

【参考代码】

```
#include <stdio.h>
#include <stdlib.h> //exit(1)所在的头文件
void main()
{
    FILE *fp, *fp1;
    char file1[20], file2[20], ch;
```

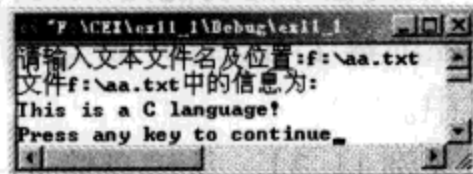


图 11.1 例 11.1 运行结果

```

printf("请输入源文本文件名及位置:");
scanf("%s", file1);
printf("请输入目标文本文件名及位置:");
scanf("%s", file2);
if( (fp = fopen(file1, "r")) == NULL) //判断文件能否打开
{
    printf("不能打开文件:%s", file1);
    exit(1);
}
fp1 = fopen(file2, "w");
printf("文件%s 中的信息为:\n", file1);
while( !feof(fp)) //若文件未结束
{
    ch = fgetc(fp);
    fputc(ch, fp1);
    printf("%c", ch); //输出字符到屏幕上
}
printf("\n");
fclose(fp); //关闭文件
fclose(fp1);

```

程序运行结果如图 11.2 所示。

本例程序的功能是从文件“f:\aa.txt”中逐个读取字符,在屏幕上显示,并将该文件中的信息写入“f:\aaa.txt”中。

【例 11.3】 从键盘输入一行字符,写入一个文件,再把该文件内容读出并显示在屏幕上。

```

#include <stdio.h>
#include <stdlib.h> //exit(1)所在的头文件
void main()
{
    FILE *fp;
    char ch;
    if( (fp = fopen("f:\\aa.txt", "wt")) == NULL) //此处一定要加双杠“\\”
    {
        printf("不能打开文件 f:\\aa.txt!");
        exit(1);
    }
    printf("请输入一串字符:\n");
    ch = getchar();
    while (ch != '\n')
    { fputc(ch, fp);

```

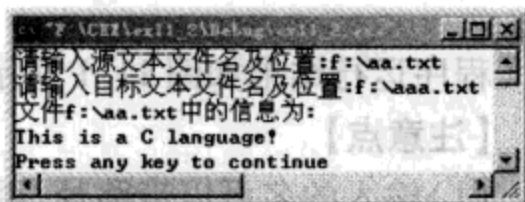


图 11.2 例 11.2 运行结果

```

    ch = getchar();
}
rewind(fp); //rewind 函数用于把 fp 所指文件的内部位置指针移到文件头
ch = fgetc(fp);
while(ch != EOF)
{ putchar(ch);
  ch = fgetc(fp);
}
printf("\n");
fclose(fp);
}

```

程序运行结果如图 11.3 所示。

【注意点】

`fp = fopen("f:\\aa.txt", "wt+")` 不能书写为: `fp = fopen("f:\\aa.txt", "wt+")`, 也就是说要有双斜杠“\\”。

3. 文件结束检测函数 `feof()` 函数

`feof()` 函数调用格式为:

`feof(文件指针);`

其功能是,判断文件内部位置指针是否处于文件结束位置,如文件结束,则返回值为 1,否则为 0。

11.2.2 字符串读写函数 `fgets()` 和 `fputs()`

1. 读字符串函数 `fgets()`

此函数的功能是从指定的文件中读一个字符串到字符数组中。函数调用的形式为:

`fgets(字符数组名或字符串指针变量, n, 文件指针);`

其功能是从文件指针所指向的文件读 $n-1$ 个字符,并将这些字符放在以字符数组名或字符串指针为起始地址的单元中。如果在读入 $n-1$ 个字符结束前遇到换行符或 EOF,读入结束,字符串读入后最后加一个“\0”字符;输入成功返回输入串的首地址;遇到文件出错返回 NULL。

例如:

```
fgets(string, n, fp);
```

其功能是从 `fp` 所指的文件中读出 $n-1$ 个字符送入字符数组 `string` 中。

【例 11.4】从 `f:\aaa.txt` 文件中读入一个含 10 个字符的字符串。

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
void main()
```

```
{ FILE *fp;
```

```
char str[11];
```

```
if((fp = fopen("f:\\aaa.txt", "rt")) == NULL)
```

```
printf("\n 不能打开文件 f:\\aaa.txt!");
```

```
exit(1);
```

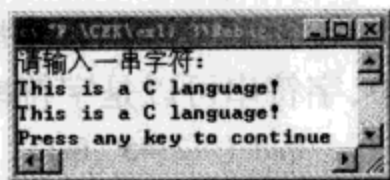


图 11.3 例 11.3 运行结果

```

    fgetc(str, 11, fp);

```

```

    printf("%s\n", str);

```

```

    fclose(fp);

```

【程序说明】

本例定义的字符数组 str 共 11 个字节,在以读文本文件方式打开文件 f:\aaa.txt 后,从中读出 10 个字符送入 str 数组,在数组最后一个单元内将加上 '\0',然后在屏幕上显示输出 str 数组。输出的 10 个字符正是例 11.2 程序的前 10 个字符。

2. 写字符串函数 fputc()

fputc() 函数的功能是向指定的文件写入一个字符串。其调用形式为:

```

fputc(字符串, 文件指针);

```

其中,字符串可以是字符串常量,也可以是字符数组名或指针变量。例如:

```

fputc("12345", fp);

```

其作用是把字符串“12345”写入 fp 所指的文件之中。

11.3 二进制文件读写

1. 数据块读写函数 fread() 和 fwrite()

C 语言还提供了用于整块数据的读写函数。可用来读写一组数据,如一个数组元素,一个结构变量的值等。

读数据块函数调用的一般形式为:

```

int fread(buffer, size, count, fp); //int 表示函数 fread() 成功读出的数据块数

```

写数据块函数调用的一般形式为:

```

int fwrite(buffer, size, count, fp); //int 表示函数 fwrite() 成功写入的数据块数

```

其中:

(1) buffer 是一个指针,在 fread() 函数中,它表示存放读入数据的首地址;在 fwrite() 函数中,它表示存放输出数据的首地址。

(2) size 表示数据块的字节数。

(3) count 表示要读写的数据块个数。

(4) fp 表示文件指针。

例如:

```

float f[2];

```

```

FILE *fp = fopen("f:\student", "r");

```

```

fread(f, 4, 2, fp);

```

其作用是从 fp 所指的文件中,每次读 4 个字节(一个 float 型实数)送入实型数组 f 中,连续读 2 次,即读 2 个实数到 f 中。

【例 11.5】从键盘输入 3 个学生数据,写入一个文件中,再读出这 3 个学生的数据并显示在屏幕上。

```

#include <stdio.h>

```

```
#include <conio.h>
struct student
{
    int id;
    char name[10];
    int age;
    float score;
} stu1[3], stu2[3], *p1, *p2;
void main()
```

```
{
    FILE *fp;
    int i;
    p1 = stu1;
    p2 = stu2;
    if((fp = fopen("f:\\student", "wb +")) == NULL)
    {
        printf("不能打开文件,按任意键退出!");
        getch(); //此函数在 conio.h 头文件中
        exit(1);
    }

    printf("输入3名学生数据:\n");
    for(i = 0; i < 3; i++, p1++)
        scanf("%d %s %d %f", &p1->id, p1->name, &p1->age, &p1->score);

    p1 = stu1;
    fwrite(p1, sizeof(struct student), 3, fp);
    rewind(fp);
    fread(p2, sizeof(struct student), 3, fp);
    printf("学号\t姓名\t年龄\t成绩\n");
    for(i = 0; i < 3; i++, p2++)
        printf("%4d\t%s\t10d%5.1f\n", p2->id, p2->name, p2->age, p2->score);
    fclose(fp);
}
```

【程序说明】

本例程序定义了一个结构 student, 定义了两个结构数组 stu1 和 stu2 以及两个结构指针变量 p1 和 p2。p1 指向 stu1, p2 指向 stu2。程序第 16 行以读写方式打开二进制文件“f:\student”, 输入 3 个学生数据之后, 写入该文件中, 然后把文件内部位置指针移到文件首, 读出 3 个学生数据后, 在屏幕上显示。

程序运行结果如图 11.4 所示。

2. 格式化读写函数 fscanf() 和 fprintf()

fscanf() 函数和 fprintf() 函数与前面使用的 scanf() 和 printf() 函数的功能相似, 都是格式化读写函数。两者的区别在于 fscanf() 函数和 fprintf() 函数的读写对象不是键盘和显示器, 而是磁盘文件。

这两个函数的调用格式为:

fscanf(文件指针, 格式字符串, 输入表列);

fprintf(文件指针,格式字符串,输出表列);

例如:

```
fscanf(fp, "%d%s", &i, s);
```

```
fprintf(fp, "%d%s", i, s);
```

用 fscanf() 和 fprintf() 函数也可以完成例 11.5 的问题。修改后的程序如例 11.6 所示。

【例 11.6】 用 fscanf() 和 fprintf() 函数完成例 11.5 的问题。



图 11.4 例 11.5 运行结果

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
#include <stdlib.h>
```

```
struct student
```

```
{
```

```
int id;
```

```
char name[10];
```

```
int age;
```

```
float score;
```

```
} stu1[3], stu2[3], *p1, *p2;
```

```
void main()
```

```
{
```

```
FILE *fp;
```

```
int i;
```

```
p1 = stu1;
```

```
p2 = stu2;
```

```
if((fp = fopen("f:\\student", "wb+")) == NULL)
```

```
{
```

```
printf("不能打开文件,按任意键退出!");
```

```
getch(); //此函数在 conio.h 头文件中
```

```
exit(1);
```

```
}
```

```
printf("输入三名学生数据:\n");
```

```
for(i=0; i<3; i++, p1++)
```

```
scanf("%d%s%d%f", &p1->id, p1->name, &p1->age, &p1->score);
```

```
p1 = stu1;
```

```
for(i=0; i<3; i++, p1++)
```

```
fprintf(fp, "%d %s %d %f\n", p1->id, p1->name, p1->age, p1->score);
```

```
rewind(fp);
```

```
for(i=0; i<3; i++, p2++)
```

```
fscanf(fp, "%d %s %d %f\n", &p2->id, p2->name, &p2->age, &p2->score);
```

```
printf("学号\t姓名\t年龄\t成绩\n");
```

```
p2 = stu2;
```

```
for(i=0; i<3; i++, p2++)
```



```
printf("%4d\t%s%6d%8.1f\n", p2->id, p2->name, p2->age, p2->score);  
fclose(fp);
```

【程序说明】

与例 11.5 相比,本程序中 `fscanf()` 和 `fprintf()` 函数每次只能读写一个结构数组元素,因此采用了循环语句来读写全部数组元素。

【注意点】

指针变量 `p1`, `p2` 是移动的,因此在程序中必须分别对它们重新赋予数组的首地址。程序运行结果如图 11.4 所示。

11.4 文件的随机读写

前面介绍的对文件的读写方式都是顺序读写,即读写文件只能从头开始,顺序读写各个数据。但在实际问题中常要求只读写文件中某一指定的部分。为了解决这个问题,可移动文件内部的位置指针到需要读写的位置,再进行读写,这种读写称为随机读写。

实现随机读写的关键是要按要求移动位置指针,这称为文件的定位。

11.4.1 文件定位

移动文件内部位置指针的函数主要有两个,即 `rewind()` 函数和 `fseek()` 函数。

1. `rewind()` 函数

前面已多次使用过 `rewind()` 函数。其调用形式为:

```
rewind(fp);
```

它的功能是把文件内部的位置指针 `fp` 移到文件首。

2. `fseek()` 函数

`fseek()` 函数用来移动文件内部位置指针。其调用形式为:

```
fseek(文件指针,偏移量,起始点);
```

其中:

(1) “偏移量”为长整型偏移量。

(2) “起始点”表示从何处开始计算位移量。规定的起始点有 3 种:文件首、当前位置和文件尾。其表示方法如表 11.2 所示。

表 11.2 `fseek()` 函数中的符号常量

符号常量	符号常量的值	含义
SEEK_SET	0	文件首
SEEK_CUR	1	当前位置
SEEK_END	2	文件末尾

例如:

```
fseek(fp, 200L, 0);
```

其作用是把位置指针移到离文件首 200 个字节处。

还要说明的是, `fseek()` 函数一般用于二进制文件。在文本文件中由于要进行字符转换, 计算出的偏移量容易产生误差, 从而导致读出的数据面目全非。

3. `ftell()` 函数

函数 `ftell()` 是获取当前位置指针函数。

其一般形式为:

`ftell(文件指针);`

功能: 得到当前文件位置指针的位置, 此位置是相对于文件头的。

返回值: 当前文件指针相对于文件头的位置。

11.4.2 文件的随机读写

在移动位置指针之后, 即可用前面介绍的任一种读写函数进行读写。由于一般是读写一个数据块, 因此常用 `fread()` 和 `fwrite()` 函数。

【例 11.7】在学生文件 `f:\student` 中用 `fwrite()` 函数写入 3 名学生信息, 并用 `fseek()` 和 `fread()` 读出第 2 名学生的信息在屏幕上显示。

【参考代码】

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct student
{
    int id;
    char name[10];
    int age;
    float score;
} stu1[3], stu, *p;
void main()
{
    FILE *fp;
    if((fp = fopen("f:\\student", "wb+")) == NULL)
    {
        printf("不能打开文件, 按任意键退出!");
        getch();
        exit(1);
    }
    printf("输入三名学生数据:\n");
    for(p = stu1; p < stu1 + 3; p++)
        scanf("%d %s %d %f", &p->id, p->name, &p->age, &p->score);
    for(p = stu1; p < stu1 + 3; p++)
        fwrite(p, sizeof(struct student), 1, fp);
    rewind(fp);
```

```

fseek(fp, sizeof(struct student), 0);
fread(&stu, sizeof(struct student), 1, fp);
printf("学号\t姓名\t年龄\t成绩\n");
printf("%4d %7s%7d %6.1f\n", stu.id, stu.name, stu.age, stu.score);
fclose(fp);
}

```

【程序说明】

- (1) 文件 student 内容与例 11.6 程序建立的文件内容无关。
- (2) 本程序用函数 fwrite() 向文件写入 3 名学生信息。
- (3) 程序中定义 stu1 为结构 student 数组, 用于存放 3 名学生信息; 定义 stu 为结构 student 变量, 用于存放读出的第 2 名学生信息; 定义 p 为指向 stu1 的指针。
- (4) 用 fseek() 函数移动文件位置指针, 从文件头开始, 移动一个 student 类型的长度, 然后用函数 fread() 读出第 2 个学生的信息。

程序运行结果如图 11.7 所示。

【注意点】

- (1) 本例使用函数 fread()、fwrite() 每次读或写一个数据块 (sizeof(struct student)) 信息。
- (2) 此例定位读取的方法如果采用例 11.6 的 student 文件中的信息, 读出的数据与实际数据将有部分不符。故采用块写函数重新写入 3 名学生的信息到文件中, 再用块读函数与定位函数读取第 2 名学生信息, 请读者自己上机体会。

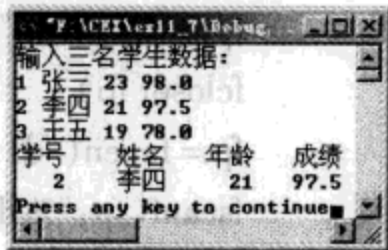


图 11.5 例 11.6 运行结果

本章小结

本章学习了 C 语言中文件操作知识, 通过多个实例, 读者应该理解文件操作的各个函数。

- (1) C 系统把文件当作一个“流”, 按字节进行处理。
- (2) C 文件按编码方式分为文本文件和二进制文件。
- (3) C 语言中, 用文件指针标识文件, 当一个文件被打开时, 可取得该文件指针。
- (4) 文件在读写之前必须打开, 读写结束必须关闭。
- (5) 文件可按只读、只写、读写、追加 4 种操作方式打开, 同时还必须指定文件的类型是二进制文件还是文本文件。
- (6) 文件可以字节、字符串、数据块为单位读写, 文件也可按指定的格式进行读写。

习题十一

一、选择题

1. 标准库函数 fgetc(s, n, f) 的功能是()。
 - A) 从文件 f 中读取长度为 n 的字符串存入指针 s 所指的内存
 - B) 从文件 f 中读取长度不超过 n-1 的字符串存入指针 s 所指的内存

C) 从文件 f 中读取 n 个字符串存入指针 s 所指的内存

D) 从文件 f 中读取长度为 n-1 的字符串存入指针 s 所指的内存

2. 有以下程序:

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
FILE *fp;
```

```
int a[10] = {1,2,3}, i, n;
```

```
fp = fopen("d1.dat", "w");
```

```
for(i=0; i<3; i++)
```

```
    fprintf(fp, "%d", a[i]);
```

```
fprintf(fp, "\n");
```

```
fclose(fp);
```

```
fp = fopen("d1.dat", "r");
```

```
fscanf(fp, "%d", &n);
```

```
printf("%d\n", n);
```

```
}
```

程序的运行结果是()。

A) 12300

B) 123

C) 1

D) 321

3. 若 fp 是指向某文件的指针, 且已读到文件末尾, 则库函数 feof(fp) 的返回值是()。

A) EOF

B) -1

C) 非零值

D) NULL

4. 在 fopen() 函数中使用文件方式“w+”, 该方式的含义是()。

A) 打开一个二进制文件读/写

B) 打开一个文本文件读/写

C) 建立一个新的文本文件读/写

D) 建立一个新的二进制文件读/写

5. 有下列程序:

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
FILE *fp;
```

```
int a[10] = {1,2,3,0,0}, i;
```

```
fp = fopen("d2.dat", "wb");
```

```
fwrite(a, sizeof(int), 5, fp);
```

```
fwrite(a, sizeof(int), 5, fp);
```

```
fclose(fp);
```

```
fp = fopen("d2.dat", "rb");
```

```
fread(a, sizeof(int), 10, fp);
```

```
fclose(fp);
```

```
for(i=0; i<10; i++)
```

```
    printf("%d", a[i]);
```

程序的运行结果是()。

- A) 1,2,3,0,0,0,0,0,0 B) 1,2,3,1,2,3,0,0,0,0
C) 123,0,0,0,0,123,0,0,0,0 D) 1,2,3,0,0,1,2,3,0,0

6. fseek() 函数的正确调用形式是()。

- A) fseek(文件指针,起始点,位移量) B) fseek(文件指针,位移量,起始点)
C) fseek(位移量,起始点,文件指针) D) fseek(起始点,位移量,文件指针)

7. 对于下述程序,在方式串分别采用“wt”和“wb”运行时,两次生成的文件 TEST 的长度是()。

```
#include <stdio.h>
```

```
void main()
```

```
{ FILE *fp = fopen("TEST","w"(或"wb"));
  fputc('A',fp);fputc('\n',fp);
  fputc('B',fp);fputc('\n',fp);
  fputc('C',fp);fclose(fp); }
```

- A) 7 字节、7 字节 B) 7 字节、5 字节 C) 5 字节、7 字节 D) 5 字节、5 字节

8. 有以下程序:

```
#include <stdio.h>
```

```
void WriteStr(char *fn,char *str)
```

```
{
  FILE *fp;
  char *s1="China", *s2="Beijing";
  fp = fopen(fn,"w");
  fputs(str,fp);
  fclose(fp); }
```

```
void main()
```

```
{
  WriteStr("t1.dat","start");
  WriteStr("t1.dat","end"); }
```

程序运行后,文件 t1.dat 中的内容是()。

- A) start B) end C) startend D) endrt

9. 有以下程序:

```
#include <stdio.h>
```

```
void main()
```

```
{
  FILE *fp; int i, k, n;
  fp = fopen("data.dat", "w+");
  for(i=1; i<6; i++)
```

```

{
    fprintf(fp, "%d ", i);
    if (i % 3 == 0) fprintf(fp, "\n");
}

```

```
rewind(fp);
```

```
fscanf(fp, "%d%d", &k, &n); printf("%d %d\n", k, n);
```

```
fclose(fp);
```

程序运行后的输出结果是()。

A) 0 0

B) 123 45

C) 1 4

D) 1 2

10. 以下叙述中错误的是()。

A) C 语言中对二进制文件的访问速度比文本文件快

B) C 语言中, 随机文件以二进制代码形式存储数据

C) 语句 `FILE fp;` 定义了一个名为 `fp` 的文件指针

D) C 语言中的文本文件以 ASCII 码形式存储数据

11. 有以下程序:

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
FILE * pf;
```

```
char * s1 = "China", * s2 = "Beijing";
```

```
pf = fopen("abc.dat", "wb+");
```

```
fwrite(s2, 7, 1, pf);
```

```
rewind(pf); /* 文件位置指针回到文件开头 */
```

```
fwrite(s1, 5, 1, pf);
```

```
fclose(pf);
```

```
}
```

程序执行后 `abc.dat` 文件的内容是()。

A) China

B) Chinang

C) ChinaBeijing

D) BeijingChina

二、填空

1. 以下程序从名为 `filea.dat` 的文本文件中逐个读入字符并在屏幕上显示, 请填空。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
FILE * fp;
```

```
char ch;
```

```
fp = fopen(____);
```

```
ch = fgetc(fp);
```

```
while(!feof(fp))
```



```

{
    putchar(ch);
    ch = fgetc(fp);
}

```

```

    putchar('\n');

```

```

    fclose(fp);
}

```

2. 以下 C 程序将磁盘中的一个文件复制到另一个文件中, 两个文件名在命令行中给出, (假定文件名无误), 请填空。

```

#include "stdio.h"
void main(int argc, char *argv[ ])
{
    FILE *f1, f2;
    char ch;
    if(argc < _____)
    {
        printf("命令行参数错!\n"); exit(0);
    }
    f1 = fopen(argv[1], "r");
    f2 = fopen(argv[2], "w");
    while(_____) fputc(fgetc(f1), _____);
    _____;
}

```

3. 以下文件用来统计文件中字符的个数, 请填空。

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
    FILE *fp;
    char ch;
    long count = 0;
    if((fp = fopen("f:\\aaa.txt", _____)) == NULL)
    {
        printf("打开文件错\n");
        exit(1);
    }
    while((ch = _____) != EOF)
    {

```

```

    count ++ ;
}
printf("count = %d\n", count);
fclose(fp);

```

4. 以下程序中函数 `huiwen()` 的功能是检查一个字符串是否是回文, 当字符串是回文时, 函数返回字符串 `yes!`, 否则函数返回字符串 `no!`, 并在主函数中输出。所谓回文即正向与反向的拼写都一样, 例如: `adgda`。请填空。

```

#include <stdio.h>
#include <string.h>
char * huiwen(char * str)
{
    char * p1, * p2;
    int i, t=0;
    p1 = str;
    p2 = _____;
    for(i=0; i <= strlen(str)/2; i++)
    if( * p1 ++ != * p2 -- )
    {
        t = 1; break;
    }
    if(_____)
    return ("yes!");
    else return ("no!");
}

void main()
{
    char str[50];
    printf("输入一串字符:");
    scanf("%s", str);
    printf("%s", _____);
}

```

三、编程题

1. 请调用 `fputs()` 函数, 把一个含有 10 个字符的字符串输入到文件中, 再从此文件中读入这 10 个字符的字符串放在一个字符串数组中, 最后把字符串数组中的字符串输出到终端屏幕, 以检验所有操作是否正确。

2. 从键盘输入 10 个浮点数, 以二进制形式存入文件中, 再从文件中读出数据并显示在屏幕上。

第12章 图书管理系统案例介绍

12.1 案例的目的和任务

本案例可作为课程项目实训内容,其目的和任务是巩固和加深学生对C语言基本知识的理解与掌握;掌握C语言编程和程序调试的基本技能;利用C语言进行基本软件设计;提高运用C语言解决实际问题的能力;锻炼书写程序设计的说明文档的能力。

12.2 案例知识要点综述

本图书管理系统主要对图书馆图书进行简单的管理,作为本课程的一个综合性案例贯穿全书大部分章节,涉及的知识点贯穿整个课程。在此案例的开发过程中,主要应用的关键知识点集中在选择结构、循环结构、函数、预处理、结构体和文件的综合应用上。

12.3 案例实训要求

- (1) 分析案例的要求。
- (2) 写出详细设计说明。
- (3) 编写参考代码,调试程序使其正确运行。
- (4) 设计完成的软件要便于操作与应用。

12.4 案例需求分析

1. 系统描述

本系统是一个虚拟的图书馆图书管理平台,功能较简单,能够实现图书的增、删、改、查;借书卡的增、删、改、查;借还书管理和查询。

2. 系统运行环境

操作系统:windows98/ME/2000/XP。

开发工具:Visual C++6.0。

12.5 案例总体设计

1. 系统功能模块图

根据案例需求分析,将本项目划分成3个总模块:图书管理、借书卡管理和借还书管理,如图12.1所示。

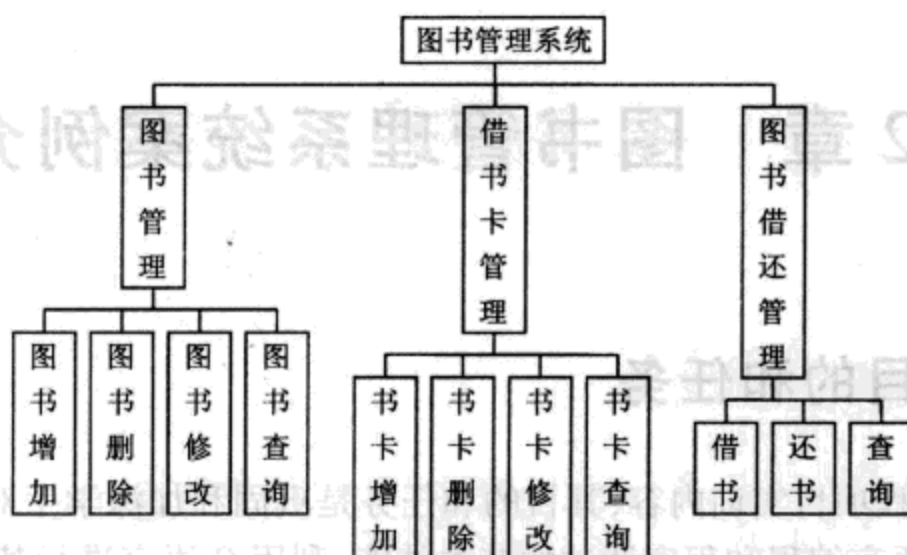


图 12.1 图书管理功能图

2. 项目数据结构

1) 图书信息数据结构设计

```

struct book //定义书的类型
{
    int booknum; //书号
    char bookname[20]; //书名
    char bookauthor[20]; //作者
    char press[50]; //出版社
    float price; //书价
    int count; //剩余本数
};
  
```

2) 借书卡信息数据结构设计

```

struct card //借书卡号
{
    int cardnum; //借书卡号
    char cardname[20]; //借书卡人姓名
    char studentorempid[20]; //学号或工号
    char memo[50]; //备注
};
  
```

3) 借还书信息数据结构设计

```

struct borrowreturn //借书卡号
{
    int cardnum; //借书卡号
    char cardname[20]; //借书卡人姓名
    int booknum; //书号
    char bookname[10]; //书名
    char bookauthor[10]; //作者
    char borr; //借或还(1 表示借,0 表示还)
    char date[11]; //借还日期
    int adminnum; //管理员号
};
  
```

12.6 案例详细设计

1. 操作界面

1) 主界面

图书管理主界面如图 12.2 所示。

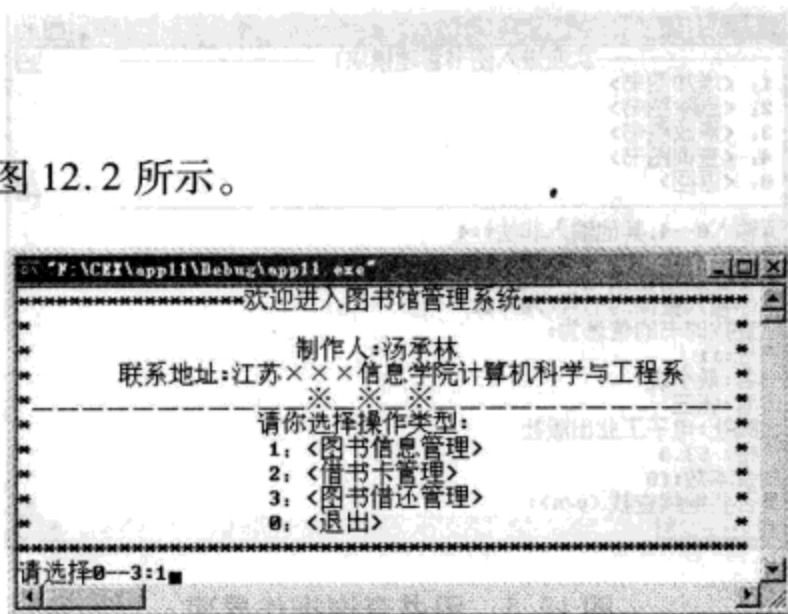


图 12.2 图书管理主界面

2) 图书增加操作界面

图书增加操作界面如图 12.3 所示。

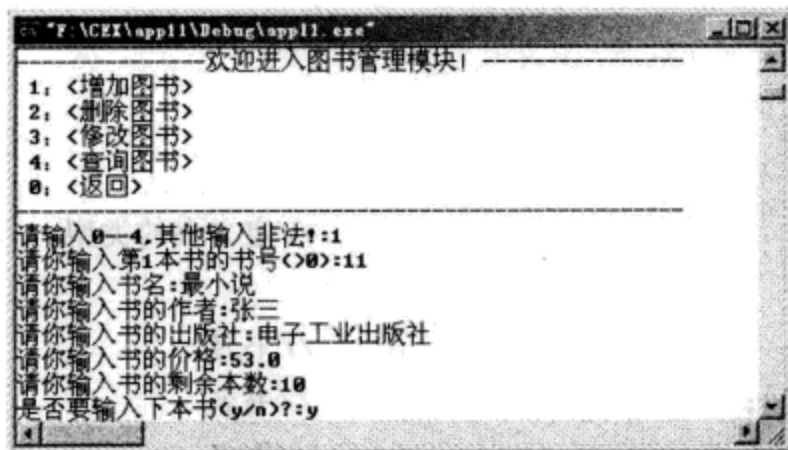


图 12.3 图书增加操作界面

3) 图书删除操作界面

图书删除操作界面如图 12.4 所示。

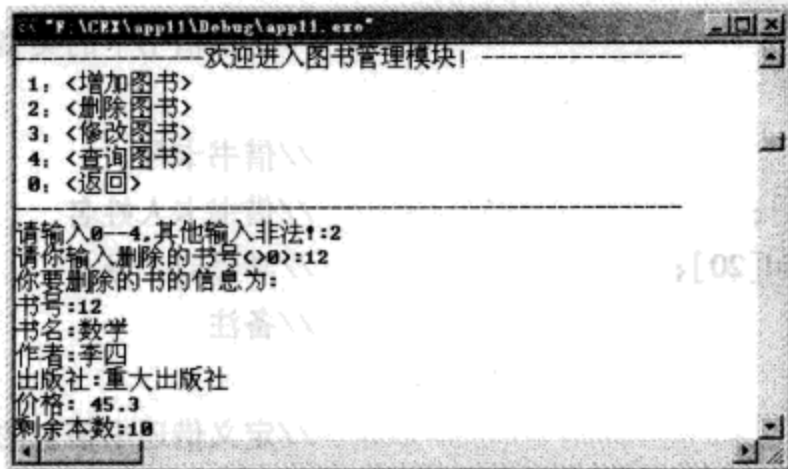


图 12.4 图书删除操作界面

4) 图书查询操作界面

图书查询操作界面如图 12.5 所示。

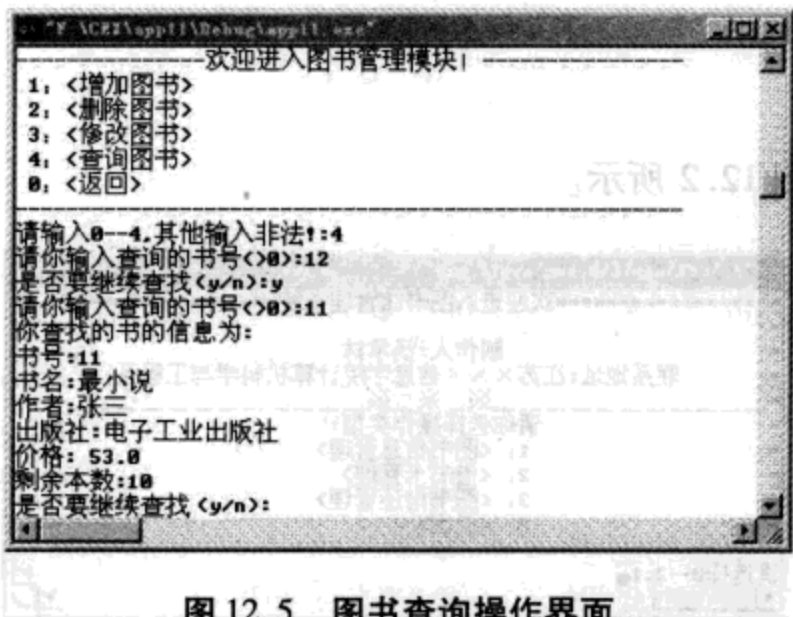


图 12.5 图书查询操作界面

限于篇幅,其他界面图形不再给出。

2. 图书管理系统源代码

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>

struct book //定义书的类型
{
    int booknum; //书号
    char bookname[20]; //书名
    char bookauthor[20]; //作者
    char press[50]; //出版社
    float price; //书价
    int count; //剩余本数
};

struct card //定义借书卡的类型
{
    int cardnum; //借书卡号
    char cardname[20]; //借书卡人姓名
    char studentorempid[20]; //学号或工号
    char memo[50]; //备注
};

struct borrowreturn //定义借还书信息的类型
{
    int cardnum; //借书卡号
    char cardname[20]; //借书卡人姓名
```



```

int booknum;           //书号
char bookname[10];     //书名
char bookauthor[10];   //作者
char borr;             //借或还(1表示借,0表示还)
char date[11];         //借还日期
int adminnum;          //管理员号
};

// ***** 图书:增加\查询\修改\删除 *****

void bookadd()          //图书增加
{
    FILE *fp;
    struct book book1;
    int i=0;
    char choice='y';
    fp=fopen("book.dat","ab+");
    while(choice=='y' || choice=='Y')
    {
        while(1)
        {
            printf("请你输入第%d本书的书号(>0):",i+1);
            scanf("%d",&book1.booknum);
            fflush(stdin);
            if(book1.booknum>0)
                break;
        }

        printf("请你输入书名:");
        scanf("%s",book1.bookname);
        fflush(stdin);
        printf("请你输入书的作者:");
        scanf("%s",book1.bookauthor);
        fflush(stdin);
        printf("请你输入书的出版社:");
        scanf("%s",book1.press);
        fflush(stdin);
        printf("请你输入书的价格:");
        scanf("%f",&book1.price);
        fflush(stdin);
        printf("请你输入书的剩余本数:");
        scanf("%d",&book1.count);
        fflush(stdin);
        fwrite(&book1,(long)sizeof(struct book),1,fp);
        fflush(stdin);
        printf("是否要输入下本书(y/n)?");
    }
}

```

```

scanf("%c",&choice);
fflush(stdin);
if(choice=='y' || choice=='Y')
{
    i++;
}
fclose(fp);
}

void booksearch() //图书查询
{
    FILE *fp;
    struct book book1;
    int num;
    char choice='y';
    fp=fopen("book.dat","rb");
    while(choice=='y' || choice=='Y')
    {
        while(1)
        {
            printf("请输入查询的书号(>0):");
            scanf("%d",&num);
            fflush(stdin);
            if(num>0)
                break;
        }
        while(!feof(fp))
        {
            fread(&book1,(long)sizeof(struct book),1,fp);
            if(book1.booknum==num)
            {
                printf("你查找的书的信息为:\n");
                printf("书号:");
                printf("%d\n",book1.booknum);
                printf("书名:");
                printf("%s\n",book1.bookname);
                printf("作者:");
                printf("%s\n",book1.bookauthor);
                printf("出版社:");
                printf("%s\n",book1.press);
                printf("价格:");
                printf("%5.1f\n",book1.price);
                printf("剩余本数:");
            }
        }
    }
}

```



```

printf("%s\n", book1. bookauthor);
printf("出版社:");
printf("%s\n", book1. press);
printf("价格:");
printf("%5.1f\n", book1. price);
printf("剩余本数:");
printf("%d\n", book1. count);
i++;
printf("你要修改的书的信息为:\n");
printf("请你输入新的书号:");
fflush(stdin);
scanf("%d", &book1. booknum);
fflush(stdin);
printf("请你输入新的书名:");
scanf("%s", book1. bookname);
fflush(stdin);
printf("请你输入新的作者:");
scanf("%s", book1. bookauthor);
fflush(stdin);
printf("请你输入新的出版社:");
scanf("%s", book1. press);
fflush(stdin);
printf("请你输入新的价格:");
scanf("%f", &book1. price);
fflush(stdin);
printf("请你输入新剩余本数:");
scanf("%d", &book1. count);
fflush(stdin);
fseek(fp, -(long) sizeof(struct book), 1);
fwrite(&book1, (long) sizeof(struct book), 1, fp);
fseek(fp, (long) sizeof(struct book), 1);
}

fflush(stdin);
printf("是否要继续修改(y/n):");
scanf("%c", &choice);
fflush(stdin);
if(choice == 'y' || choice == 'Y')
{
    rewind(fp); i = 0;
}
fclose(fp);

```

```

}
void bookdelete()
{
    FILE *fp, *fp1;
    struct book book1;
    int num; int i = 0;
    fp = fopen("book.dat", "rb");
    fp1 = fopen("bookbak.dat", "wb");
    while(1)
    {
        printf("请输入删除的书号(>0):");
        scanf("%d", &num);
        fflush(stdin);
        if(num > 0)
            break;

        while(!feof(fp))
        {
            i = 0;
            fread(&book1, (long)sizeof(struct book), 1, fp);
            if(book1.booknum == num && i == 0)
            {
                printf("你要删除的书的信息为:\n");
                printf("书号:");
                printf("%d\n", book1.booknum);
                printf("书名:");
                printf("%s\n", book1.bookname);
                printf("作者:");
                printf("%s\n", book1.bookauthor);
                printf("出版社:");
                printf("%s\n", book1.press);
                printf("价格:");
                printf("%5.1f\n", book1.price);
                printf("剩余本数:");
                printf("%d\n", book1.count);
                i++;
            }
            else
                fwrite(&book1, (long)sizeof(struct book), 1, fp1);
        }
        fclose(fp);
        fclose(fp1);
    }
}

```

```

fp = fopen("book.dat", "wb");
fp1 = fopen("bookbak.dat", "rb");
while( !feof( fp1 ) )
{
    fread( &book1, (long) sizeof( struct book ), 1, fp1 );
    fwrite( &book1, (long) sizeof( struct book ), 1, fp );
}
fclose( fp );
fclose( fp1 );
getche(); //暂停一下, 否则返回菜单太快!
fflush( stdin );

// ***** 结束图书操作: 增加\查询\修改\删除 *****
// ***** 借书卡: 增加\查询\修改\删除 *****

void cardadd( ) //借书卡增加
{
    FILE * fp;
    struct card card1;
    int i = 0;
    char choice = 'y';
    fp = fopen( "card.dat", "wb + " );
    while( choice == 'y' || choice == 'Y' )
    {
        while( 1 )
        {
            printf( "请输入第 %d 张借书卡号 ( > 0 ) : ", i + 1 );
            scanf( "%d", &card1. cardnum );
            fflush( stdin );
            if( card1. cardnum > 0 )
                break;
        }

        printf( "请输入借书卡人姓名: " );
        scanf( "%s", card1. cardname );
        fflush( stdin );
        printf( "请输入借书卡人学号或工号: " );
        scanf( "%s", card1. studentorempid );
        fflush( stdin );
        printf( "请输入借书卡的备注: " );
        scanf( "%s", card1. memo );
        fflush( stdin );
        fwrite( &card1, (long) sizeof( struct card ), 1, fp );
        fflush( stdin );
    }
}

```



```

printf("是否要输入下一张借书卡信息(y/n)?");
scanf("%c",&choice);
fflush(stdin);
if(choice == 'y' || choice == 'Y')
    i++;
}
fclose(fp);

void cardsearch() //借书卡查询
{
    FILE *fp;
    struct card card1;
    int num;
    char choice = 'y';
    fp = fopen("card.dat","rb");
    while(choice == 'y' || choice == 'Y')
    {
        while(1)
        {
            printf("请输入查询的借书卡号(>0):");
            scanf("%d",&num);
            fflush(stdin);
            if(num > 0) break;
        }

        while(!feof(fp))
        {
            fread(&card1,(long)sizeof(struct card),1,fp);
            if(card1.cardnum == num)
            {
                printf("你查找的借书卡的信息为:\n");
                printf("借书卡号:");
                printf("%d\n",card1.cardnum);
                printf("借书卡人姓名:");
                printf("%s\n",card1.cardname);
                printf("借书卡学号或工号:");
                printf("%s\n",card1.studentorempid);
                printf("借书卡备注:");
                printf("%s\n",card1.memo);
                break;
            }
        }

        printf("是否要继续查找(y/n):");
    }
}

```

```

scanf("%c",&choice);
fflush(stdin);
if(choice == 'y' || choice == 'Y')
{
    rewind(fp);
}
fclose(fp);
}

void cardupdate() //借书卡修改
{
    FILE *fp;
    struct card card1;
    int num;int i;
    char choice = 'y';
    fp = fopen("card.dat","rb+");
    while(choice == 'y' || choice == 'Y')
    {
        while(1)
        {
            printf("请输入修改的借书卡号(>0):");
            scanf("%d",&num);
            if(num > 0) break;
            fflush(stdin);
            if(num < 0) break;
        }
        i = 0;
        fread(&card1,(long)sizeof(struct card),1,fp);
        if(card1.cardnum == num && i == 0) //i == 0 的条件是使如下信息显示一次
        {
            printf("你要修改的借书卡信息为:\n");
            printf("卡号:");
            printf("%d\n",card1.cardnum);
            printf("姓名:");
            printf("%s\n",card1.cardname);
            printf("学号或工号:");
            printf("%s\n",card1.studentorempid);
            printf("备注:");
            printf("%s\n",card1.memo);
            i++;
            printf("你要修改的借书卡信息为:\n");
            printf("请输入新的借书卡号(>0):");
            scanf("%d",&card1.cardnum);
            fflush(stdin);
        }
    }
}

```

```

printf("请输入新的借书卡人姓名:");
scanf("%s", card1. cardname);
fflush( stdin );
printf("请输入新的借书卡人学号或工号:");
scanf("%s", card1. studentorempid);
fflush( stdin );
printf("请输入新的借书卡备注:");
scanf("%s", card1. memo);
fflush( stdin );
fseek( fp, - (long) sizeof( struct card ), 1 );
fwrite( &card1, (long) sizeof( struct card ), 1, fp );
fseek( fp, (long) sizeof( struct card ), 1 );

}

fflush( stdin );
printf("是否要继续修改借书卡信息(y/n):");
scanf("%c", &choice);
fflush( stdin );
if( choice == 'y' || choice == 'Y' )
    rewind( fp );
}

fclose( fp );

void carddelete() //借书卡删除
{
    FILE * fp, * fp1;
    struct card card1;
    int num; int i = 0;
    fp = fopen("card. dat", "rb");
    fp1 = fopen("cardbak. dat", "wb");
    while( 1 )
    {
        printf("请输入删除的借书卡号( >0 ):");
        scanf("%d", &num);
        if( num > 0 ) break;
    }
    fflush( stdin );
    while( !feof( fp ) )
    {
        i = 0;
        fread( &card1, (long) sizeof( struct card ), 1, fp );
        if( card1. cardnum == num && i == 0 )

```

```

    {
        printf("你要删除的借书卡信息为:\n");
        printf("借书卡号:");
        printf("%d\n", card1.cardnum);
        printf("借书卡人姓名:");
        printf("%s\n", card1.cardname);
        printf("借书卡人学号或工号:");
        printf("%s\n", card1.studentorempid);
        printf("借书卡备注:");
        printf("%s\n", card1.memo);
        i++;
    }
    else
        fwrite(&card1, (long) sizeof(struct card), 1, fp1);
}

fclose(fp);
fclose(fp1);
fp = fopen("card.dat", "wb");
fp1 = fopen("cardbak.dat", "rb");
while(!feof(fp1))
{
    fread(&card1, (long) sizeof(struct card), 1, fp1);
    fwrite(&card1, (long) sizeof(struct card), 1, fp);
}

fclose(fp);
fclose(fp1);
getche(); //暂停一下, 否则返回菜单太快!
fflush(stdin);

// ***** 结束借书卡操作: 增加\查询\修改\删除 *****
// ***** 借还书 *****
void bookborrow() //借书
{
    FILE *fp, *fp1, *fp2;
    struct card card1;
    struct book book1;
    struct borrowreturn borrowreturn1;
    int num = 0, num1 = 0; int i = 0;
    char choice = 'y', choice1 = 'n', choice2 = 'n';
    /* choice 表示是否继续查找借书卡号, choice1 是否放弃查询借书卡或书号,
       choice2 表示是否真的借书 */
    fp = fopen("card.dat", "rb");
    fp1 = fopen("book.dat", "rb");

```

```

fp2 = fopen("borrowreturn.dat", "a+");
// ***** 查询借书卡号 *****
while( choice == 'y' || choice == 'Y' )
{
    printf("请你输入借阅者的借书卡号(>0):");
    scanf("%d", &num);
    fflush(stdin);
    while( !feof(fp) )
    {
        fread( &card1, (long) sizeof( struct card ), 1, fp );
        if( card1.cardnum == num )
        {
            printf("你查找的借书卡的信息为(>0):\n");
            printf("借书卡号:");
            printf("%d\n", card1.cardnum);
            printf("借书卡人姓名:");
            printf("%s\n", card1.cardname);
            printf("借书卡学号或工号:");
            printf("%s\n", card1.studentorempid);
            printf("借书卡备注:");
            printf("%s\n", card1.memo);
            break;
        }
    }
    if( num == 0 )
    {
        printf("没有找到你找的借卡号(>0)!");
        printf("放弃借书吗(y/n):");
        scanf("%c", &choice1);
        fflush(stdin);
        if( choice1 == 'y' || choice1 == 'Y' )
            exit(0);
    }
    printf("是否要继续查找吗(y/n):");
    scanf("%c", &choice);
    fflush(stdin);
    if( choice == 'y' || choice == 'Y' )
    {
        rewind(fp); // i = 0;
    }
}
fclose(fp);

```

```

// ***** 查询借阅书号 *****
choice = 'y';
while( choice == 'y' || choice == 'Y')
{
    printf("请你输入借阅的书号(>0):");
    scanf("%d",&num1);
    fflush(stdin);
    while( !feof(fp1))
    {
        fread( &book1, (long) sizeof( struct book), 1, fp1 );
        if( book1.booknum == num1 )
        {
            printf("你要借阅的书的信息为:\n");
            printf("书号:");
            printf("%d\n",book1.booknum);
            printf("书名:");
            printf("%s\n",book1.bookname);
            printf("作者:");
            printf("%s\n",book1.bookauthor);
            printf("出版社:");
            printf("%s\n",book1.press);
            printf("价格:");
            printf("%5.1f\n",book1.price);
            printf("剩余本数:");
            printf("%d\n",book1.count);
            break;
        }
        if( num1 == 0 || book1.count == 0 )
        {
            printf("没有找到你找的书号或此书已借阅完!");
            printf("放弃借书吗(y/n):");
            scanf("%c",&choice1);
            fflush(stdin);
            if( choice1 == 'y' || choice1 == 'Y')
            {
                exit(0);
            }
        }
    }
    printf("是否要继续查找(y/n):");
    scanf("%c",&choice);
    fflush(stdin);
    if( choice == 'y' || choice == 'Y')

```



```

    rewind(fp1); //i=0;
}

// ***** 保存借阅信息 *****
printf("真的借书吗(y/n):");
scanf("%c",&choice2);
fflush(stdin);
if(!(choice2=='y' || choice2=='Y'))
{
    exit(0);
}
else
{
    if(!(card1.cardnum!=0 && book1.booknum!=0 && book1.count!=0))
    {
        exit(0);
    }
    //card1.cardnum 写入的借书卡号, book1.booknum 写入的借书书号
    book1.count--; //修改所借书的剩余本数
    fseek(fp1, -(long)sizeof(struct book), 1); //定位文件内部指针到修改的信息位置
    fwrite(&book1, (long)sizeof(struct book), 1, fp1); //修改所借书的剩余本数
    borrowreturn1.cardnum = card1.cardnum;
    strcpy(borrowreturn1.cardname, card1.cardname);
    borrowreturn1.booknum = book1.booknum;
    strcpy(borrowreturn1.bookname, book1.bookname);
    strcpy(borrowreturn1.bookauthor, book1.bookauthor);
    borrowreturn1.borr = '1';
    printf("请输入管理员号:");
    scanf("%d",&borrowreturn1.adminnum);
    fflush(stdin);
    printf("请输入借书日期(格式要求:2009-02-06):");
    scanf("%s",&borrowreturn1.date);
    fflush(stdin);
    fwrite(&borrowreturn1, (long)sizeof(struct borrowreturn), 1, fp2);

    fclose(fp1);
    fclose(fp2);
}

void bookreturn() //还书
{
    FILE *fp, *fp1, *fp2;
    struct card card1;

```

```

struct book book1;
struct borrowreturn borrowreturn1;
int num = 0, num1 = 0; int i = 0;
char choice = 'y', choice1 = 'n', choice2 = 'n';
/* choice 表示是否继续查找还书卡号, choice1 是否放弃查询还书卡或书号,
choice2 表示是否真的还书 */
fp = fopen("card. dat", "rb");
fp1 = fopen("book. dat", "rb +");
fp2 = fopen("borrowreturn. dat", "ab +");
// ***** 查询还书者的借书卡号 *****
while( choice == 'y' || choice == 'Y')
{
    printf("请输入还书者的借书卡号 (>0):");
    scanf("%d", &num);
    fflush(stdin);
    while( !feof(fp) )
    {
        fread( &card1, (long) sizeof( struct card ), 1, fp);
        if( card1. cardnum == num)
        {
            printf("你查找的还书者的借书卡的信息为:\n");
            printf("还书人的借书卡号:");
            printf("%d\n", card1. cardnum);
            printf("还书人姓名:");
            printf("%s\n", card1. cardname);
            printf("还书人学号或工号:");
            printf("%s\n", card1. studentorempid);
            printf("还书人借书卡备注:");
            printf("%s\n", card1. memo);
            break;
        }
    }
    if( num == 0)
    {
        printf("没有找到你找的借书卡号!");
        printf("放弃还书吗(y/n):");
        scanf("%c", &choice1);
        fflush(stdin);
        if( choice1 == 'y' || choice1 == 'Y')
        {
            exit(0);
        }
    }
}

```

```

printf("是否要继续查找(y/n):");
scanf("%c",&choice);
fflush(stdin);
if(choice == 'y' || choice == 'Y')
{
    rewind(fp); //i=0;
}
fclose(fp);
// ***** 查询还书书号 *****
choice = 'y';
while(choice == 'y' || choice == 'Y')
{
    printf("请你输入还书的书号(>0):");
    scanf("%d",&num1);
    fflush(stdin);
    while(!feof(fp1))
    {
        fread(&book1,(long)sizeof(struct book),1,fp1);
        if(book1.booknum == num1)
        {
            printf("你要还的书的信息为:\n");
            printf("书号:");
            printf("%d\n",book1.booknum);
            printf("书名:");
            printf("%s\n",book1.bookname);
            printf("作者:");
            printf("%s\n",book1.bookauthor);
            printf("出版社:");
            printf("%s\n",book1.press);
            printf("价格:");
            printf("%5.1f\n",book1.price);
            printf("剩余本数:");
            printf("%d\n",book1.count);
            break;
        }
        if(num1 == 0)
        {
            printf("没有找到你找的书号!");
            printf("放弃还书吗(y/n):");
            scanf("%c",&choice1);
            fflush(stdin);

```

```

        if( choice1 == 'y' || choice1 == 'Y')
        {
            exit(0);
        }
    }

    printf("是否要继续查找(y/n):");
    scanf("%c",&choice);
    fflush(stdin);
    if( choice == 'y' || choice == 'Y')
    {
        ***** 查找待还书 *****
        rewind(fp1); //i = 0;
        while( choice == 'y' || choice == 'Y')
        {
            printf("请输入要还书的卡号(>0):");
            // ***** 保存还书信息 *****
            printf("真的还书吗(y/n):");
            scanf("%c",&choice2);
            fflush(stdin);
            if( !( choice2 == 'y' || choice2 == 'Y') )
            {
                exit(0);
            }
            else
            {
                if( !( card1.cardnum != 0 && book1.booknum != 0 ) )
                {
                    exit(0);
                }
                //card1.cardnum 写入的还书人的借书卡号
                //book1.booknum 写入的借书书号
                book1.count ++; //修改所还书的剩余本数
                fseek( fp1, -(long) sizeof( struct book ), 1 ); //定位文件内部指针到修改的信息位置
                fwrite( &book1, (long) sizeof( struct book ), 1, fp1 ); //修改所还书的剩余本数
                borrowreturn1.cardnum = card1.cardnum;
                strcpy( borrowreturn1.cardname, card1.cardname );
                borrowreturn1.booknum = book1.booknum;
                strcpy( borrowreturn1.bookname, book1.bookname );
                strcpy( borrowreturn1.bookauthor, book1.bookauthor );
                borrowreturn1.borr = '0'; //0 表示还书
                printf("请输入管理员号:");
                scanf("%d",&borrowreturn1.adminnum);
                fflush(stdin);
                printf("请输入还书日期(格式要求:2009-02-06):");
                scanf("%s",&borrowreturn1.date);
            }
        }
    }
}

```

```

fflush( stdin );
fwrite( &borrowreturn1, (long) sizeof( struct borrowreturn ), 1, fp2 );

}

fclose( fp1 );
fclose( fp2 );

}

void bookborr( ) //借还书查询
{
    FILE * fp;
    struct borrowreturn borrowreturn1;
    int num;
    char choice = 'y';
    fp = fopen( "borrowreturn. dat", "rb" );

    while( choice == 'y' || choice == 'Y' )
    {
        printf( "请输入查询的借书卡号(>0):" );
        scanf( "%d", &num );
        fflush( stdin );

        printf( "你查询的借还书信息为:\n" );
        printf( " 卡号 借书卡人姓名 书号 作者 出版社 借/还 借/还日期 管理员号\n" );

        while( !feof( fp ) )
        {
            if( fread( &borrowreturn1, (long) sizeof( struct borrowreturn ), 1, fp ) != 1 )
            {
                break; //保证文件中最后一条记录不会重复显示
            }

            if( borrowreturn1. cardnum == num )
            {
                printf( "%6d ", borrowreturn1. cardnum );
                printf( "%10s ", borrowreturn1. cardname );
                printf( "%6d ", borrowreturn1. booknum );
                printf( "%10s ", borrowreturn1. bookname );
                printf( "%10s ", borrowreturn1. bookauthor );
                printf( "%6s ", ( borrowreturn1. borr == '1' ) ? "借" : "还" );
                printf( "%13s ", borrowreturn1. date );
                printf( "%6d\n", borrowreturn1. adminnum );

                printf( "是否要继续查找(y/n):" );
                scanf( "%c", &choice );
                fflush( stdin );
            }
        }
    }
}

```

```

        if( choice == 'y' || choice == 'Y')
        {
            rewind( fp );
        }
    }
    fclose( fp );

// *****结束借还书 *****
void main()
{
    char ch1, ch2, ch3, ch4;
    do
    {
        printf( " ***** 欢迎进入图书馆管理系统 *****\n" );
        printf( " *                               * \n" );
        printf( " *                               * \n" );
        printf( " *      联系地址:江苏 × × × 信息学院计算机科学与工程系      * \n" );
        printf( " * _____*_*_*_____ * \n" );
        printf( " *                               * \n" );
        printf( " *      请你选择操作类型:                               * \n" );
        printf( " *      1: <图书信息管理>                               * \n" );
        printf( " *      2: <借书卡管理>                               * \n" );
        printf( " *      3: <图书借还管理>                               * \n" );
        printf( " *      0: <退出>                                       * \n" );
        printf( " ***** \n" );
        printf( "请选择 0 -- 3:" );
        scanf( "%c", &ch1 );
        getchar( );
        switch( ch1 )
        {
            case '1':
                do
                {
                    printf( "----- 欢迎进入图书管理模块!----- \n" );
                    printf( " 1: <增加图书> \n" );
                    printf( " 2: <删除图书> \n" );
                    printf( " 3: <修改图书> \n" );
                    printf( " 4: <查询图书> \n" );
                    printf( " 0: <返回> \n" );
                    printf( "----- \n" );
                    printf( "请输入 0 -- 4, 其他输入非法!:" );
                    scanf( "%c", &ch2 );
                    getchar( );

```



```

switch( ch2)
{
    case '1':bookadd(); break;
    case '2':bookdelete();break;
    case '3':bookupdate();break;
    case '4':booksearch();break;
    case '0':break;
}
} while( ch2 != '0');break;
case '2':
do
{
    printf("----- 欢迎进入借书卡管理模块!----- \n");
    printf(" 1: <增加借书卡> \n");
    printf(" 2: <删除借书卡> \n");
    printf(" 3: <修改借书卡> \n");
    printf(" 4: <查询借书卡> \n");
    printf(" 0: <返回> \n");
    printf("----- \n");
    printf("请输入 0 --4,其他输入非法!");
    scanf("%c",&ch3);
    getchar();
    switch( ch3)
    {
        case '1':cardadd(); break;
        case '2':carddelete();break;
        case '3':cardupdate();break;
        case '4':cardsearch();break;
        case '0':break;
    }
} while( ch3 != '0');break;
case '3':
do
{
    printf("----- 欢迎进入借还书管理模块!----- \n");
    printf(" 1: <借书> \n");
    printf(" 2: <还书> \n");
    printf(" 3: <借还书查询> \n");
    printf(" 0: <返回> \n");
    printf("----- \n");
    printf("请输入 0 --3,其他输入非法!");
    scanf("%c",&ch4);

```

```

getchar();
switch( ch4)
{
    case '1':bookborrow(); break;
    case '2':bookreturn(); break;
    case '3':bookborr(); break;
    case '0':break;
}
while( ch4 != '0');break;
case '0':exit(0);
}
while( ch1 != '0');
}

```

12.7 案例总结

本案例实现了图书管理系统中对图书和借书卡信息的增、删、改、查,以及借还书管理和查询等基本操作。根据软件基本流程,对各个模块的设计过程作了简要的阐述。

本案例对图书信息的管理采用文件的形式,限于篇幅且由于 C 语言进行图形界面设计较复杂,本系统只提供了一些简洁的提示信息,不支持图形用户界面,因而阅读不太方便。读者可利用 C 语言所提供的图形处理功能,把一些界面设计得漂亮一些,另外可根据需要完善系统的各种功能。

实 验

实验一 熟悉 Visual C ++ 6.0 环境

实验目的

- (1) 熟悉 PC 机的配置环境。
- (2) 熟悉 Visual C ++ 6.0 环境及 C 语言源程序的编辑、编译、连接和执行操作。
- (3) 理解 C 程序的基本结构。

第一部分 教师指导

本实验着重编写一个简单的 C 语言小程序,熟悉一下 Visual C ++ 6.0 的基本操作。

练习 1:熟悉 Visual C ++ 6.0 环境

【解题思路】

- (1) 开始→程序→Microsoft Visual C ++ 6.0。
- (2) 文件→新建→win32 console application→工程名称→Hello→确定。
- (3) 文件→新建→文件→C ++ source file→文件名→hello. c→确定。
- (4) 输入如下代码

```
#include <stdio. h >
```

```
void main( )
```

```
{
```

```
printf("hello world!\n");
```

- (5) 组建→编译[hello. c](编译过程,生成 hello. obj 文件)。
- (6) 组建→组建[hello. exe](连接过程,生成 hello. exe 文件)。
- (7) 组建→执行[hello. exe](执行过程,执行 hello. exe 文件)。

练习 2:查看下面程序的输出结果。

```
#include <stdio. h >
```

```
void main( )
```

```
{
```

```
printf("How are you!\n");
```

```
}
```

【思考】

- (1) 如果去掉语句“printf("How are you!\n");”中的分号“;”,结果如何? 写出运行结果。
- (2) 如果把“printf("How are you!\n");”改为“printf("How\tare\tyou!\n");”,结果又如何?

【问题分析】

- (1) 提示如下信息:

F:\CEX\hello\hello. c(5) : error C2143: syntax error : missing ';' before '}'

执行 cl. exe 时出错。

hello.obj - 1 error(s), 0 warning(s)

(2) 运行结果: How are you! (两个单词词首相距 8 个字符)。

第二部分 练习

编写一个简单程序, 实现如下输出结果:

Joan: Hello!

Mary: Hello!

Joan: How old are you?

Mary: I am eighteen.

第三部分 作业

编写一个简单程序, 用汉字输出三行信息: 一行输出自己的学号; 一行输出自己的姓名; 一行输出自己的联系地址。

第四部分 思考、总结及书写实验报告

实验二 数据类型及运算符

实验目的

- (1) 学会正确定义变量的类型。
- (2) 掌握各种运算与表达式的书写格式。
- (3) 掌握简单 C 语言程序(顺序结构)语句的功能及书写格式。
- (4) 掌握 scanf() 与 printf() 函数的使用方法。

第一部分 教师指导

练习 1: 互换两个变量的值

【解题思路】

接收两个整数, 分别保存在两个变量中, 通过第三个变量将这两个变量的值互换。程序需要三个整型变量, 前两个变量用来存储用户输入的两个整数, 第三个变量用来作中间变量, 借助于这个中间变量, 将前面两个变量的值互换。

【参考代码】

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int a,b,c;
```

```
printf("\n 请输入第一个数:");
```

```
scanf("%d",&a);
```

```
printf("\n 请输入第二个数:");
```

```
scanf("%d",&b);
```

```
/* 显示互换前的数 */
```

```
printf("\n\n 输出互换前的数:");
```

```
printf("第一个数是:%d",a);
```

```
printf("第二个数是:%d",b);
```

```
/* 互换两个数 */
```

```
c = a;
```

```
a = b;
```

```
b = c;
```

```
/* 显示互换后的数 */
```

```
printf("\n\n 输出互换后的数:");
```

```
printf("现在的第一个数是:%d",a);
```

```
printf("现在的第二个数是:%d",b);
```

```
}
```

练习 2: 写出下列程序的运行结果。

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int a,b;
```

```
a = 2;
```

```
a * = 3 + 7;
```

```
printf("%d,%o,%x\n",a,a,a);
```

```
a * = b = 5;
```

```
printf("%4d,%4o,%4X\n",a,a,a);
```

```
a% = (b% = 2);
```

```
printf("a = %d,b = %d\n",a,b);
```

```
a + = b - = a * = a;
```

```
printf("a = %d,b = %d\n",a,b);
```

```
}
```

【思考】 如果去掉“a% = (b% = 2);”中的“()”,是否影响结果?

第二部分 练习

练习 3: x,y,z 为三个整数,编写程序实现输出下列每步的 x,y,z 的值。

(1) x = y = z = 1;

```
z = x ++;
```

(2) z = (++x) * (--y);

(3) z + = - (x ++) + (++y);

(4) z + = (x ++) + (y --);

第三部分 作业

编写一个程序,接收一个四位整数,将该数的每一位数字相加并显示结果。

第四部分 思考、总结及书写实验报告

实验三 输入/输出语句

实验目的

- (1) 掌握 C 语言输入/输出的方法。
- (2) 掌握 C 语言中字符的输入/输出方法。
- (3) 学会编写简单的 C 语言程序。
- (4) 掌握 scanf() 与 printf() 函数的使用方法。

第一部分 教师指导

练习 1: 输入下列源程序:

```
#include <stdio.h>

void main()
{
    int a,b;
    float c,d;
    char c1,c2;
    double e,f;
    long m,n;
    unsigned int p,q;
    a=97;b=98;
    c1='A';c2='B';
    c=3.14;d=-6.7;
    e=123.56032;f=0.987654321;
    m=65536;n=-65538;
    p=123456;q=987654;
    printf("a=%d,b=%d\nc1=%c,c2=%c\nc=%f,d=%6.2f\n",a,b,c1,c2,c,d);
    printf("e=%15le,f=%15.4le\nm=%ld,n=%ld\np=%u,q=%u\n",e,f,m,n,p,q);
}
```

(1) 运行此程序并分析结果(如图 lab.1)。

(2) 改用 scanf() 函数输入数据而不用赋值语句,其形式如下:

```
scanf("%d,%d,%c,%c,%f,%f,%lf,%lf,%ld,%ld,%u,%u",&a,&b,&c1,&c2,&c,&d,&e,&f,&m,&n,&p,&q);
```

输入如下数据:

97,98,A,B,3.14,-6.98,1223.4322,0.9876333,
888888,-999999,543332,56777 回车

分析运行结果。

(3) 在(2)的基础上将 printf() 语句改为:

```
printf("a=%o,b=%O\nc1=%x,c2=%X\nc=%15.6e,d=%15.5e\n",a,b,c1,c2,c,d);
printf("e=%15lf,f=%15.4lf\nm=%ld,n=%ld\np=%lo,q=%lX\n",e,f,m,n,p,q);
```

观察并分析结果。

练习 2: 用 getchar() 函数输入三个字符,然后用 printf() 函数按输入次序输出这三个字符,并输出每个字符的 ASCII 值,最后用 putchar() 函数按相反的顺序输出这三个字符。

【解题思路】

定义三个字符变量 c1,c2,c3,用于接收三个字符,用 printf() 函数与 putchar() 函数输出。

【参考代码】

```
#include <stdio.h>

void main()
{
```

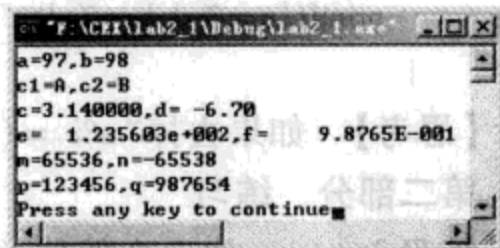


图 lab.1


```
char c1,c2,c3;
printf("请输入第一个字符:");
c1 = getchar();
fflush(stdin); //用于清除键盘缓冲区
printf("请输入第二个字符:");
c2 = getchar();
fflush(stdin);
printf("请输入第三个字符:");
c3 = getchar();
fflush(stdin);
printf("输出的三个字符:\n");
printf("%c,%d\n%c,%d\n%c,%d\n",c1,c1,c2,c2,c3,c3);
putchar(c1);putchar('\n');
putchar(c2);putchar('\n');
putchar(c3);putchar('\n');
```

【思考】

- (1)省略三个“putchar('\n');”语句,输出结果如何?
- (2)如果不定义三个变量,又如何?
- (3)没有语句“fflush(stdin);”又如何? 如果换成“getchar()”,能行吗?

第二部分 练习

练习3:接收一个字符,例如用“+”表示加法,用“-”表示减法。接收 num1 和 num2 两个数。如果输入的字符为“+”,则将两个数相加并显示相应的结果;如果输入的字符为“-”,则将这两个数相减并显示相应的值。

第三部分 作业

编写一个程序,要求根据用户输入的长和宽,计算矩形的面积和周长。

第四部分 思考、总结及书写实验报告

实验四 选择结构

实验目的

- (1)熟练掌握 if 语句的运用。
- (2)熟练掌握 if-else 语句及其嵌套语句。
- (3)掌握 switch 语句。

第一部分 教师指导

练习1:给出一个百分制成绩,要求输出成绩等级 A、B、C、D、E。90 分以上为 A,80~90 分为 B,70~79 分为 C,60~69 分为 D,60 分以下为 E。要求用:(1)if 语句;(2)if-else 语句;(3)switch 语句。

【解题思路】

用 scanf() 输入 1 个数。

- (1)用 5 个 if 语句判断实现;

(2) 用 4 个 if-else 语句嵌套实现;

(3) 把输入的数转换成整数, 即 11 个等级(0,1,...,10), 然后用 switch 语句实现。

【参考代码】

(1) if 语句

```
#include <stdio.h>

void main()
{
    float score;
    printf("输入一个成绩:");          //提示行
    scanf("%f",&score);
    printf("该成绩等级:");            //提示行
    if(score >= 90)
        printf("A");
    if(score >= 80 && score < 90)
        printf("B");
    if(score >= 70 && score < 80)
        printf("C");
    if(score >= 60 && score < 70)
        printf("D");
    if(score < 60)
        printf("E");
    printf("\n");
}
```

(2) if-else 语句

```
#include <stdio.h>

void main()
{
    float score;
    printf("输入一个成绩:");
    scanf("%f",&score);
    printf("该成绩等级:");
    if(score >= 90)
        printf("A");
    else if(score >= 80)
        printf("B");
    else if(score >= 70)
        printf("C");
    else if(score >= 60)
        printf("D");
    else
        printf("E");
    printf("\n");
}
```

(3) switch 语句

```
#include <stdio.h>
```

```
void main()
```

```
{  
    float score;
```

```
    int c;
```

```
    printf("输入一个成绩:");
```

```
    scanf("%f",&score);
```

```
    c = (int)(score/10);
```

//等级转换

```
    switch(c)
```

```
{  
    case 0:
```

```
    case 1:
```

```
    case 2:
```

```
    case 3:
```

```
    case 4:
```

```
    case 5: printf("该成绩等级:"); printf("E"); break;
```

```
    case 6: printf("该成绩等级:"); printf("D"); break;
```

```
    case 7: printf("该成绩等级:"); printf("C"); break;
```

```
    case 8: printf("该成绩等级:"); printf("B"); break;
```

```
    case 9:
```

```
    case 10: printf("该成绩等级:"); printf("A"); break;
```

```
    default: printf("输入数据不正确!\n");
```

```
    printf("\n");
```

//把系统输出串“press any key to continue”移到下一行

第二部分 练习

练习 2:企业年底发放的奖金数额根据当年的利润提成。情况确定:利润低于或等于 10 万元时,奖金可提 10%;利润高于 10 万元,低于 20 万元时,低于 10 万元部分按 10% 提成,高于 10 万元部分可提成 7.5%;利润在 20 万元到 40 万元之间时,高于 20 万元的部分,可提成 5%;利润在 40 万元到 60 万元之间时,高于 40 万元部分,可提成 3%;利润在 60 到 100 万元之间时,高于 60 万元的部分,可提成 1.5%;利润高于 100 万元时,超过 100 万元部分按 1% 提成。输入当年的利润,求应发放的奖金总数。要求用 if 语句、if-else 语句、switch 语句三种之一实现。

第三部分 作业

编写一个程序,计算长方形、圆形和三角形的面积,根据用户的选择计算相应形状的面积。要求用:(1)if 语句;(2)if-else 语句;(3)switch 语句。

第四部分 思考、总结及书写实验报告

实验五 循环结构(一)

实验目的

(1)掌握 for 循环语句。

(2) 熟悉 break 和 continue 语句在循环语句中的使用。

(3) 掌握通过嵌套循环解决实际问题。

第一部分 教师指导

练习 1: 编写一个程序, 用于接收用户输入的 10 个字符, 统计其中的大写字母和小写字母的个数, 并显示相应的信息。

【解题思路】

此程序需要 3 个整型变量和一个字符型变量。程序的输入将存储在字符型变量中。一个整型变量为 for 循环的计数器, 统计接收的字符数, 直到字符的个数达到 10, 循环结束; 另外两个整型变量用作小写字母和大写字母的计数器。用户可输入任何字符, 程序将利用输入字符的 ASCII 值检查这些字符是大写字母还是小写字母。

【参考代码】

```
#include <stdio.h>
void main()
{
    char instring;
    int i, small, big;           // 统计大(小)写字母的计数器是 big, small
    printf("\n 请输入一串字符:");
    small = 0; big = 0;
    for (i = 0; i < 10; i++)
    {
        instring = getchar();
        if ( instring >= 'a' && instring <= 'z')
            small++;
        else if( instring >= 'A' && instring <= 'Z')
            big++;
    }
    for (i = 0; i < 10; i++)
        putchar(instring);
    printf("small = %d 个, big = %d 个\n", small, big);
}
```

练习 2: 编写一个程序, 生成如图 lab. 2 所示的输出结果, 要求根据用户输入的一个大写字母, 输出一个由字母组成的图案。

请输入一个数: E

EEEE

DDDD

CCC

BB

A

a

bb

ccc

dddd

eeee

图 lab. 2

【解题思路】

- (1) 将图案分成上下两个部分, 分别打印。
- (2) 使用嵌套循环, 外层循环控制打印的行数, 内层循环控制该行打印的字符及打印的个数。

【参考代码】

```
#include <stdio.h>
void main()
{
    int i, j, k;           //i 用于控制行数, j 用于控制列数
    char letter;
    printf("请输入一个大写字母:");
    scanf("%c", &letter);
    k = letter - 'A';       //k 表示输入字母与'A'的差, 用于下三角形的行数控制
    if (!(letter >= 'A' && letter <= 'Z'))
        printf("输入的不是大写字母!");
    else
    {
        for (i = letter - 'A'; i >= 0; i--)
        {
            printf("\n");
            for (j = 0; j <= i; j++)
                printf("%c", letter);
            letter--;
            for (i = 0; i <= k; i++)
            {
                printf("\n");
                for (j = 0; j <= i; j++)
                    printf("%c", 'a' + i);
            }
            printf("\n");
        }
    }
}
```

第二部分 练习

练习 3: 改造练习 2 程序, 输出如图 lab. 3 的图案。

```
EEEEEEEEEE
DDDDDDDD
CCCCC
BBB
A
a
bbb
cccc
ddddddd
eeeeeeeeee
```

图 lab. 3

练习 4:编写一个程序,最多接收 10 个整数,求出其中所有正整数的积。用户可以通过输入 9999 终止程序,统计用户输入的正数的个数,并显示这些正数的积。

【提示】

如果输入的数为负,则忽略该数,不做累积,然后接收下一个数(使用 continue 语句)。

第三部分 作业

打印所有的“水仙花数”。所谓“水仙花数”是指一个三位数,其各位数字立方和等于该数本身。例如: $153 = 1^3 + 5^3 + 3^3$ 。

【提示】 以 999 终止循环。

第四部分 思考、总结及书写实验报告

实验六 循环结构(二)

实验目的

- (1)掌握使用 while 语句、do-while 语句实现循环的方法。
- (2)掌握使用 while 循环、do-while 循环及嵌套循环编写的程序。
- (3)掌握 break 语句、continue 语句的使用。

第一部分 教师指导

练习 1:编写一个程序,让用户输入一个正整数值,然后计算各位数的积。例如,如果输入的数是 123,则计算结果是 6。若输入 504,则输出应该是 0。

【解题思路】

需要使用嵌套循环,内层循环控制显示输出结果,外层循环控制用户是否愿意继续,使用一个字符型变量存放用户对问题“您是否要继续(y/n)”的回答,如果输入 y,则继续,否则退出程序。

定义一个变量 x 接收输入的数据,y 用于保存输入的原始数据,每次内循环 x 的值是上次循环 x 整除 10 后的值,这里的主要运算是整除和求余。mult 作为累积器,其初值为 1。

```
mult * = x % 10;
```

```
x /= 10;
```

内层循环结束条件为 $x = 0$ 。外层循环结束条件为非'y'(或非'Y')。

【参考代码】

```
#include <stdio.h>
void main()
{
    char yesno;
    int x, mult, y;
    yesno = 'y';          /* 该值必须初始化 */
    while(yesno == 'y' || yesno == 'Y')
    {
        mult = 1;
        printf("请输入一个数:");
        scanf("%d", &x);
        y = x;            /* 保存输入数 */
    }
}
```



```

while (x)                                /* 该循环用于输出字符 */
{
    mult *= x%10;                        /* 求各位上数字之积 */
    x /= 10;                             /* 右移一位 */
    printf("%d 各位上的数字之积:%d\n", y, mult);
    printf("您是否要继续:");
    fflush(stdin);                       /* 清空键盘缓冲区 */
    scanf("%c", &yesno);                /* 用户的选择是否(y/n)继续 */
}

```

练习 2: 将一个正整数分解质因数。例如: 输入 40, 打印出 $40 = 2 * 2 * 2 * 5$ 。

【解题思路】

对 n 进行分解质因数, 应先找到一个最小的质数 i , i 值应该从 2 开始, 然后按下述步骤完成:

(1) 如果这个质数恰等于 n , 则说明分解质因数的过程已经结束, 打印出即可;

(2) 如果 $n \neq i$, 但 n 能被 i 整除, 则应打印出 i 的值, 并用 n 除以 i 的商, 作为新的一正整数 n , 重复执行第一步;

(3) 如果 n 不能被 i 整除, 则用 $i+1$ 作为 i 的值, 重复执行第一步。

【参考代码】

```

#include <stdio.h>
void main()
{
    int n, i;
    printf("请输入一个数:");
    scanf("%d", &n);
    printf("%d = ", n);
    i = 2;
    while(i <= n)                        // 注意循环次数, n 在不断地变化
    {
        while(n != i)                    // n != i 说明 i 继续加 1 直到等于 n
        {
            if(n % i == 0)                // 判断 n 能否被 i 整除
            {
                printf("%d * ", i);
                n = n / i;                // 求 n/i 的商赋给 n
            }
            else
                break;                    // n 不能被 i 整除, 跳出内循环
        }
        i++;
    }
}

```

```
printf("%d\n", n); //输入最后的商 n
```

第二部分 练习

练习 3:编写一个程序,根据输入计算圆的面积、矩形面积和三角形面积。输入界面如图 lab. 4 所示。最后询问用户是否继续,直到用户自己确定要退出程序。

第三部分 作业

改造练习 1,使用 do-while 循环实现计算正整数各位数的积。

第四部分 思考、总结及书写实验报告

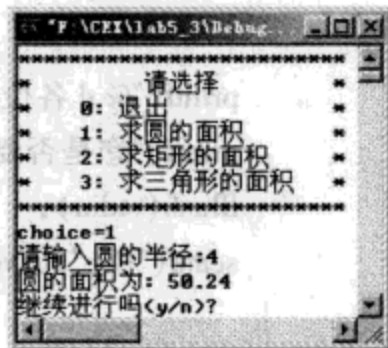


图 lab. 4

实验七 数组(一)

实验目的

- (1) 通过实验,加深对一维数组、下标变量的理解,学会正确使用数组编写程序。
- (2) 掌握一维数组元素的赋值、求和、比较、交换和排序的方法。

第一部分 教师指导

练习 1:输入 10 个整数,将其按由小到大排序。要求采用冒泡法。

【解题思路】

排序是程序设计中经常遇到的问题,其中冒泡排序是一种行之有效的方法。设有 5 个元素 $a[0], \dots, a[4]$, 要求经过排序后,使 $a[0], \dots, a[4]$ 由小到大按序排列。其基本思路是将数组中两相邻元素 $a[i], a[i+1]$ 进行比较,将小的放到 $a[i]$ 中,大的放在 $a[i+1]$ 中。当所有元素都比较完后,最大的元素将成为数组中最后一个元素 $a[4]$,然后再将前面的 $a[0], \dots, a[3]$ 两两比较。按同样道理,得到 $a[3]$ 为 4 个元素中的最大元素,依次得到 $a[2], \dots, a[0]$, 排序完成。例如,有 5 个数为:5, 3, 4, 2, 1, 请将其按由小到大顺序排列。

开始:[5, 3, 4, 2, 1]

第 1 次循环:[3, 4, 2, 1], 5

第 2 次循环:[3, 2, 1], 4, 5

第 3 次循环:[2, 1], 3, 4, 5

第 4 次循环:[1], 2, 3, 4, 5

注意:“[]”内数的变化。

【参考代码】

```
#include <stdio.h>
#define N 10
void main()
{
    int a[10];
    int i, j, t;
    printf("请输入 10 个数:\n");
    for(i = 0; i < N; i++)
        scanf("%d", &a[i]);
    for(i = 0; i < N - 1; i++)
```

第二部分 练习

【解题思路】

例如:要把下列数据按升序排序,则直接插入排序过程如下,其中括号中的数表示已排好序。

初始数据: [5] 3 4 1 2

第1步插入 [3 5] 4 1 2

第2步插入 [3 4 5] 1 2

第3步插入 [1 3 4 5] 中2

第4步插入 [1 2 3 4 5]

第三部分 作业

【提示】

(1) 定义数组 $a[11]$ (已有有序元素 10 个), 插入的元素 x 。

(2) 首先考虑插入的数与数组的第 $a[0]$ 和第 $a[9]$ 比较, 插入到相应位置。

(3) 再与 $a[i]$ 及 $a[i+1]$ 比较(i 从 0 到 9), 如果介于两者之间, 则将从 $i+1$ 位置的元素向后移, 在第 $i+1$ 位置插入 x , 之后要使用 `break` 语句跳出循环。

2. 有以下程序,其功能是读入 10 个整数,统计负数的个数,并计算负数之和,请填空。

```
#include <stdio.h>

void main()
{
    int a[10], sum = 0, i, j = 0;
    for(i = 0; i < 10; i++)
        scanf("%d", &a[i]);
    for(i = 0; i < 10; i++)
    {
        if(a[i] > 0)
            _____;
        sum += a[i];
        _____;
    }
    printf("负数个数:%d, 负数之和:%d\n", j, sum);
}
```

第四部分 思考、总结及书写实验报告

实验八 数组(二)

实验目的

- (1) 掌握二维数组元素的赋值、求和、比较、交换和排序的方法。
- (2) 掌握字符数组的字符串复制、连接和比较的方法,并掌握求字符串长度的方法。

第一部分 教师指导

练习 1: 输入三个字符串,找出其中最大者。

【解题思路】

有一个二维的字符数组 str, 大小为 3 行 20 列, 每一行可容纳 20 个字符。可把 str[0], str[1], str[2] 看作三个一维字符数组, 可以用 gets() 函数分别读入三个字符串。经过两次比较, 就可以得到最大者。把它放在一维数组 string 中。

【参考代码】

```
#include <stdio.h>
#include <string.h>

void main()
{
    char string[20];
    char str[3][20];
    int i;
    printf("请输入三个字符串:\n");
    for(i = 0; i < 3; i++)
        gets(str[i]);
    if(strcmp(str[0], str[1]) > 0)
        strcpy(string, str[0]);
}
```

```

else
    strcpy(string, str[1]);
if( strcmp(str[2], string) > 0)
    strcpy(string, str[2]);
printf("三个字符串最大者: %s\n", string);

```

练习 2: 输入三名学生的三门课程成绩, 要求计算每名同学的成绩总分, 并输出每门课程的平均分。

【解题思路】

定义一个 4 行 4 列的一个二维数组, 每行前三个元素存放每名同学的三门课程成绩, 每行最后一个元素存放每名同学的总分, 第四行的前三个元素存放每门课程的平均分。

【参考代码】

```

#include <stdio.h>

void main()
{
    float a[4][4] = {{0}, {0}, {0}, {0}};
    int i, j;
    printf("请输入三名学生三门课程的成绩:\n");
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            scanf("%f", &a[i][j]);
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
        {
            a[i][3] += a[i][j];
            a[3][j] += a[i][j];
        }
    for(j=0; j<3; j++)
        a[3][j] = a[3][j]/3;
    printf("      成绩1   成绩2   成绩3   总分\n");
    for(i=0; i<4; i++)
    {
        if(i<3)
            printf("      ");
        else
            printf("平均分");
        for(j=0; j<4; j++)
            printf("%6.1f", a[i][j]);
        printf("\n");
    }
}

```

程序运行结果如图 lab. 5 所示。



图 lab.5

第二部分 练习

练习 3: 输入三个字符串, 按由小到大排序。

第三部分 作业

设计一个学生成绩统计程序, 基本要求如下:

- (1) 要求输入的信息含有学号(学号从 1 到 30, 程序自动产生)和成绩;
- (2) 计算平均分;
- (3) 统计各区间的人数, 其区间分为 90 ~ 100、80 ~ 89、70 ~ 79、60 ~ 69、60 分以下;
- (4) 计算大于等于平均分和小于平均分的人数。

第四部分 思考、总结及书写实验报告

实验九 函数(一)

实验目的

- (1) 掌握使用不带参数的用户自定义函数的定义和调用方法。
- (2) 掌握使用带参数的用户自定义函数的定义和调用方法。

第一部分 教师指导

练习 1: 编写二个函数, 用于接收三角形的底和高, 并计算三角形的面积。要求:

- (1) 一个函数中输入三角形的底和高, 不将面积作为返回值, 在子函数中输出三角形的面积;
- (2) 一个函数接收主函数中的三角形的底和高, 将面积作为返回值返回, 在 main() 函数中调用求面积的函数, 并接收函数的返回值, 将求得的面积值输出。

【解题思路】

本例较简单, (1) 题中要求在主函数输入三角形的底和高, 当然要定义两个变量接收三角形的底和高了。

(2) 题不需要传递三角形的底和高, 在子函数中要建立两个变量, 用于接收输入的三角形的底和高, 并处理、输出三角形的面积。

【参考代码】

```
#include <stdio.h>

void main()
{
    float fun1(float, float);
    void fun2();
```



```
float x,y;
printf("请输入三角形的底和高 x,y:\n");
scanf("%f%f",&x,&y);
printf("调用函数 fun1() 求三角形的面积:%5.2f\n",fun1(x,y));
fun2();
}

float fun1(float x,float y)
{
    return(x*y/2);
}

void fun2()
{
    float x,y;
    printf("请输入三角形的底和高 x,y:\n");
    scanf("%f%f",&x,&y);
    printf("调用函数 fun2() 求三角形的面积:%5.2f\n",x*y/2);
}
```

练习 2: 编写一个函数 fun(), 函数的功能是根据分段函数

$$y = \begin{cases} x & x < 1 \\ 2x - 1 & 1 \leq x < 10 \\ 3x - 11 & x \geq 10 \end{cases}$$

求变量 x 对应的 y 值, y 值由函数值返回。例如: 当 x 输入 3, y 的输出为 5。请补全函数。

【解题思路】

本题检查数学函数的编程及分支语句的嵌套, 本题在用 if-else 语句的嵌套形式表示分段函数时, 一嵌套要正确, 二表达式书写要正确。

```
#include <stdio.h>
#include <conio.h>
int fun(int x)
{
    //该函数位于 conio.h 头文件中

    printf("请输入 x:");
    scanf("%d",&x);
    printf("y = %5d\n",fun(x));
}
```

【参考代码】

```
int fun(int x)
{
    void main()
```

```

int y;
if(x < 1)
    y = x;
else if(x >= 1 && x < 10)
    y = 2 * x - 1;
else
    y = 3 * x - 11;
return y;
}

```

第二部分 练习

练习 3: 编写函数 fun() 实现计算 n!。要求主函数中实现数据输入, 函数 fun() 返回 n!。

例如: 输入为 5, 函数返回值为 120。

```

#include <stdio.h>

long fun(int n)
{
    // 计算 n! 的函数
    // ... (此处省略部分代码) ...
}

void main()
{
    int n;
    long f;
    printf("输入 n:");
    scanf("%d", &n);
    f = fun(n);
    printf("n! = : %ld\n", f);
}

```

练习 4: 请编写函数 fun(), 其功能是: 将两个两位数的正整数 a、b 合并形成一个整数放在 c 中。合并的方式是将 a 数的十位和个位数依次放在 c 数个位和十位上, b 数的十位和个位数依次放在 c 数的百位和千位上。

例如, 当 a = 16, b = 35, 调用该函数后, c = 5361。

注意: 部分源程序给出如下。请勿改动主函数 main() 和其他函数中的任何内容, 仅在函数 fun() 的花括号中填入所编写的若干语句。

程序:

```

#include <stdlib.h>
#include <stdio.h>
long fun(int a, int b)
{
    // 合并两个两位数的函数
    // ... (此处省略部分代码) ...
}

void main()
{
    // 主函数
    // ... (此处省略部分代码) ...
}

```

```

{
    int a,b;
    long c;
    printf("输入 a,b:");
    scanf("%d%d",&a,&b);
    c = fun(a,b);
    printf("The result is:%ld\n", c);
}

```

第三部分 作业

编写函数 fun(), 它的功能是求 n 以内(不包括 n)同时能被 5 与 11 整除的所有自然数之平方根的和 s, 并作为函数值返回。

例如: n 为 1000 时, 函数值应为 s = 391.834424。

注意: 部分源程序给出如下。请勿改动主函数 main() 和其他函数中的任何内容, 仅在函数 fun() 的花括号中填入所编写的若干语句。

程序:

```

#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
double fun(int n)
{
    // 在此处编写代码

}

void main()
{
    system("CLS");
    printf("s = %f\n", fun(1000));
}

```

【提示】 本题的解题思路是逐个取得从 0 ~ n 之间的所有数, 对每次取得的数进行条件判断, 条件是既能被 5 整除同时又能被 11 整除, 注意: 这两个条件要求同时成立, 因此用到了“&&”运算符。满足条件, 该数平方根就被累加到 s 中去。

第四部分 思考、总结及书写实验报告

实验十 函数(二)

实验目的

- (1) 掌握数组作为函数参数的传递方法。
- (2) 掌握字符数组作为函数参数的传递方法。

第一部分 教师指导

练习 1: 输入数组, 将数组中的最小值与第一个元素交换, 最大值与最后一个元素交换, 输

出数组。

【解题思路】

用函数实现“最大值与最后一个元素交换,最小值与第一个元素交换”。用数组作为参数,相当于采用引用方式,函数对参数中元素的修改影响实参的值。函数实现:通过一个循环对比找到最大值和最小值所在的下标值,然后通过一个临时变量将最大值与最后一个元素交换,最小值与第一个元素交换。

【参考代码】

```
#include <stdio.h>

void max_min(int a[5]);
void main()
{
    int b[5];
    int i;
    printf("请输入 5 个数组元素:\n");
    for(i=0;i<5;i++)
        scanf("%d",&b[i]);
    max_min(b);
    for(i=0;i<5;i++)
        printf("%d ",b[i]);
    printf("\n");
}

void max_min(int a[5])
{
    int i,k,n;          //k 用来记录最大值的下标,n 用来记录最小值的下标
    int;
    int max,min;
    max = min = a[0];
    k = n = 0;
    for(i=1;i<5;i++)
    {
        if(max < a[i])
        {
            k = i;
            max = a[i];
        }
        else if(min > a[i])
        {
            n = i;
            min = a[i];
        }
    }
    t = a[0];           //将最小值和第一个元素互换
    a[0] = a[n];
    a[n] = t;
    t = a[k];           //将最大值和最后一个元素互换
    a[k] = a[4];
    a[4] = t;
}
```

```

a[n] = t;
t = a[4]; //将最大值和最后一个元素互换
a[4] = a[k];
a[k] = t;
}

```

练习 2: 编写一个函数 fun(), 将输入的国家名按字母顺序排序。请补全函数。

```

#include <stdio.h>
#include <string.h>
#define N 5
void fun(char str[N][20])
{
}

void main()
{

```

```

    char str[N][20];
    int i;
    printf("\n 请输入%d个国家(地区)的名称:\n", N);
    for(i=0; i<N; i++)
        gets(str[i]);
    fun(str);
    printf("按字母顺序排列如下:\n");
    for(i=0; i<N; i++)
        printf("%s\n", str[i]);
}

```

【解题思路】

输入的内容是一系列国家名, 需要按照字母顺序对其进行排序。程序将接收用户的输入, 并将这些国家保存在一个字符串数组中, 使用 strcmp() 函数比较两个字符串的大小, 使用 strcpy() 函数将一个字符串拷贝到另一个字符串中。

【参考代码】

```

void fun(char str[N][20])
{
    int i, j;
    char s[20];
    for (i=0; i<N-1; i++)
        for (j=0; j<N-1-i; j++)
            if (strcmp(str[j], str[j+1])>0)
            {
                strcpy(s, str[j]);
                strcpy(str[j], str[j+1]);
                strcpy(str[j+1], s);
            }
}

```

第二部分 练习

练习 3: 给定程序 MOD11.C 中函数 fun() 的功能是计算 n 门课程的平均分, 计算结果作为函数值返回。例如, 若有 10 门课程的成绩是: 81, 61.5, 98, 78.5, 64, 71, 77, 65, 76.5, 91, 则函数的返回值为 76.35。请补全函数。

```
#include <stdio.h>
float fun(float score[], int n)
{
    // 补全函数
}

void main()
{
    float score[10] = {81, 61.5, 98, 78.5, 64, 71, 77, 65, 76.5, 91}, aver;
    aver = fun(score, 10);
    printf("平均成绩: %5.2f\n", aver);
}
```

第三部分 作业

MOD11.C 函数 fun() 的功能是先将字符串 s 中的字符按逆序存放到 t 中, 然后把 s 中的字符按正序连接到 t 串的后面。例如: 当 s 中的字符串为“12345”时, 则 t 中的字符串应为“5432112345”。请补全函数。

```
#include <stdio.h>
#include <string.h>
void fun(char s[], char t[])
{
    // 补全函数
}

void main()
{
    char s[80], t[80];
    printf("请输入字符串 s:");
    scanf("%s", s);
    fun(s, t);
    printf("字符串 t: %s\n", t);
}
```

第四部分 思考、总结及书写实验报告

实验十一 指针(一)

实验目的

- (1) 掌握指针变量的定义和引用。
- (2) 掌握指针与变量、指针与数组、指针与字符串的关系。
- (3) 掌握指针作为函数参数的方法。
- (4) 重点掌握指向一维数组的指针的用法。

第一部分 教师指导

练习 1:从键盘输入 5 个整数存入一个数组,用指针的方法从中查找某个整数,找到(可以有多个,以第 1 个为准)时输出该数所在的下标,否则输出消息:“没有找到”,输出结果请参考图 lab. 6。

请输入 5 个整数:13 35 57 76 55
请输入要查找的数:57
该数所在的位置为:2

图 lab. 6 练习 1 的输出结果

【解题思路】

定义一个数组 `a[5]` 用来存放 5 个输入的数,一个指针变量用来指向数组 `a`,一个 `flag` 标志,找到了设为 1,没有找到设为 0。

【参考代码】

```
#include <stdio.h>
void main()
{
    int a[5], *p, x, n, flag = 0;
    n = 0;
    printf("请输入 5 个整数:\n");
    for(p = a; p < a + 5; p++) /* 此循环用于接收值 */
        scanf("%d", p);
    printf("\n 请输入要查找的数:");
    scanf("%d", &x);
    for(p = a; p < a + 5; p++)
    {
        if(*p == x)
        {
            flag = 1;
            break;
        }
        n++;
    }
    if(flag == 0)
        printf("没有找到!");
    else
        printf("该数所在的下标为 %d \n", n);
}
```

【思考】

直接用数组名 `a` 和下标变量编写程序,实现本题功能。

练习 2:用指针编写一个求字符串长度的函数。

【解题思路】

定义一个字符数组,把字符数组名作为函数的实参传递给函数,以指针变量作为形参,在

函数中求出字符串长度,并返回字符串长度。

【参考代码】

```
#include <stdio.h>

int fun(char *p)
{
    int i=0;
    while( *p != '\0')
    { i++; p++; }
    return i;
}

void main()
{ char str[80];
  printf("请输入字符串:");
  gets(str);
  printf("字符串 str 的长度为:%d\n",fun(str));
}
```

第二部分 练习

练习 3:将数组中的数按逆序存放,要求用指针变量编写程序。

【提示】 假设数组 $a[N]$ 中有 N 个元素,例如,有 10 个元素的数组 $a[10]$ 的数据:10,20,30,40,50,60,70,80,90,100,逆序的方法是将第一个数 10 与最后一个数 100 交换,再将第二个数 20 与倒数第二个数 90 交换,以此类推,共交换 5 次后即可完成重新存放。现设计两个指针 $p1$ 和 $p2$, $p1$ 指向第一个数组元素, $p2$ 指向最后一个元素,然后 $p1$ 和 $p2$ 所指的元素交换,每交换一次, $p1$ 向后移一个元素, $p2$ 向前移一个元素,直到 $p1 \geq p2$ (或 $p1 \geq a + N/2$) 为止。

练习 4:将字符数组中的字符按逆序存放,要求用指针变量编写程序。

【提示】 对练习 3 作简单的改动即可实现。

第三部分 作业

1. 输入一行文字,找出其中大写字母、小写字母、空格、数字及其他字符各有多少? 要求用指针变量实现。

2. 编写一个 C 程序,用于接收两个数组的值,将这两个数组中的值依次相加保存在第三个数组中,要求用指向数组的指针实现。

第四部分 思考、总结及书写实验报告

实验十二 指针(二)

实验目的

- (1) 掌握指针变量的定义和引用。
- (2) 掌握指针与变量、指针与数组、指针与字符串的关系。
- (3) 掌握指针作为函数参数的方法。
- (4) 重点掌握指向二维数组的指针的用法。

第一部分 教师指导

练习1: 写出一个函数, 将一个 3×3 的矩形转置。要求使用: (1) 指向数组的指针变量; (2) 使用指针数组; (3) 用指向 n 个元素的一维数组指针变量。

【解题思路】

(1) 假设数组为 $a[3][3]$, 指针变量为 p , p 指向数组 a , 则 $*(p+3*i+j)$ 表示第 i 行 j 列的元素, $*p$ 是 $a[0][0]$, 而 $*(p+1)$ 则是 $a[0][1]$, 而不是 $a[1][0]$ 。

(2) 定义一个指针数组: $\text{int } *p[3]$, 给指针数组赋初值: $p[i] = a[i]$ ($i=0,1,2$)。

(3) 定义一个指向 n 个元素的一维数组指针变量: $\text{int } (*p)[3]$, 注意“ $[]$ ”中的 3 是指一维数组长度为 3 。用 $*(*(p+i)+j)$ 表示第 i 行第 j 列元素。

【参考代码】

方法一: 使用指向数组的指针变量实现

```
#include <stdio.h>

void fun(int *p)                //形参是指针变量
{
    int i, j, t;
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
        {
            t = *(p+3*i+j);      // p+3*i+j 第 i 行第 j 列元素的地址
            *(p+3*i+j) = *(p+3*j+i);
            *(p+3*j+i) = t;
        }
}

void main()
{
    int a[3][3], *p, i, j;
    printf("请输入 3 行 3 列的矩阵:\n");
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            scanf("%d", &a[i][j]);
    p = &a[0][0];
    fun(p);                      //指向二维数组的指针变量作实参
    printf("转置后 3 行 3 列的矩阵:\n");
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            printf("%4d", a[i][j]);
    printf("\n");
}
```

程序运行结果如图 lab.7 所示。

方法二: 使用指针数组实现

```
#include <stdio.h>

void fun(int *p[3])            //形参必须是指针数组
```

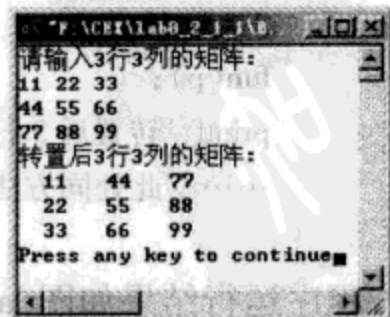


图 lab.7

```

    int i, j, t;
    for(i=0; i<3; i++)
        for(j=i; j<3; j++)
        {
            t = *(p[i] + j);
            *(p[i] + j) = *(p[j] + i);
            *(p[j] + i) = t;
        }
}

void main()
{
    int a[3][3], *p[3], i, j;
    .....//此处同方法一
    for(i=0; i<3; i++)          //给指针数组赋初值
        p[i] = a[i];
    fun(p);                      //调用函数 fun(), 指针数组名作实参
    .....//此处同方法一
}

```

程序运行结果如图 lab. 7 所示。

方法三: 使用指向 n 个元素的指针变量实现

```

#include <stdio.h>

void fun(int (*p)[3])          //形参必须是指向 3 个元素的一维数组指针变量
{
    int i, j, t;
    for(i=0; i<3; i++)
        for(j=i; j<3; j++)
        {
            t = *( *(p+i) + j);
            *( *(p+i) + j) = *( *(p+j) + i);
            *( *(p+j) + i) = t;
        }
}

void main()
{
    int a[3][3], (*p)[3], i, j;
    .....//此处同方法一
    p = a;                      //给 p 赋初值
    fun(p);                     //调用函数 fun(), 指针数组名作实参
    printf("转置后 3 行 3 列的矩阵:\n");
    .....//此处同方法一
}

```

程序运行结果如图 lab. 7 所示。

第二部分 练习

练习 2: 请编写函数 fun(), 函数的功能是将放在字符串数组中的 M 个字符串(每串长度

不超过 N),按顺序合并成一个新的字符串。例如,字符串数组的 M 个字符串如下:

AAAA

BBBBBBBBBB

CCC

则合并后的字符串的内容是 AAAABBBBBBBBBBBCCC。请补全函数。

要求采用:(1)数组和指针变量;(2)指针数组和指针变量;(3)指向 n 个元素的一维数组指针变量。

```
#include <stdio.h>
#include <string.h>
#define M 3
#define N 20
void fun(char w[M][N],char *b)
```

```
{
void main()
```

```
{
char a[M][N] = {"AAAA","BBBBBBBBBB","CCC"};
char str[80] = "\0";
fun(a,str);
printf("字符串 str:%s\n",str);
```

【提示】

(1)定义一个二维字符数组 a[M][N],一个存放连接后的字符数组 str[80],使用字符串连接函数 strcat(字符串 1,字符串 2),把字符串 2 连接到字符串 1 后面,并在字符串 1 后面自动加上结束标志'\0'。

(2)使用指针数组:char * p[M],用 p[i] = a[i] (i = 0,1,……,M - 1)给指针数组赋初值。

(2)本题还可以使用 char (*p)[N]进行赋值,赋值要求移动指针,p 称指向 N 个元素的一维数组的指针变量,p + i 指向 a[i]行。

第三部分 作业

编写一个函数 fun(),用于将输入的国家名按字母顺序排序。请补全函数,要求使用指针数组实现。

```
#include <stdio.h>
#include <string.h>
#define N 5
void fun(char *p[20])
{
}
void main()
```

```

char str[N][20], *p[20];
int i;
printf("请输入%d个国家(地区)的名称:\n", N);
for(i=0; i<N; i++)
    gets(str[i]);
for(i=0; i<N; i++)
    p[i] = str[i]; //给指针数组赋初值
fun(p);
printf("按字母顺序排列如下:\n");
for(i=0; i<N; i++)
    printf("%s\n", p[i]); //p[i]可以用str[i]替换

```

第四部分 思考、总结及书写实验报告

实验十三 结构与共用(一)

实验目的

- (1) 掌握结构类型的概念和定义方法以及结构变量的定义与引用。
- (2) 掌握使用结构数组的方法。
- (3) 理解结构作为函数参数。

第一部分 教师指导

练习 1: 创建结构。编写一个程序,用于创建一个结构 goods,用来存储商品信息:商品编号、商品名称、单价和数量,接收用户输入的值并显示这些值。

goods 结构成员类型如表 lab. 1 所示。

表 lab. 1

商品编号	商品名称	单价	数量
6 个字符	20 个字符	单精度浮点数	整型

【解题思路】

结构的成员包括商品编号、商品名称、单价和数量。商品编号和商品名称用字符表示,单价为浮点型,而数量是整型。

【参考代码】

```

#include <stdio.h>
/* goods 结构 */
struct goods
{
    char id[6];
    char name[25];
    float price;
    int number;

```



```

};
void main()
{
    struct goods goods1;           // 结构变量
    /* 用结构变量接收用户输入的值 */
    printf("**** 请输入商品的详细信息 **** \n");
    printf("商品编号:");
    scanf("%s", goods1.id);        // 或用 gets(goods1.id);
    fflush(stdin);                  // 清除键盘缓冲区信息
    printf("商品名称:");
    scanf("%s", goods1.name);      // gets(goods1.name);
    printf("单价:");
    scanf("%f", &goods1.price);
    printf("数量:");
    scanf("%d", &goods1.number);
    /* 用结构变量接收用户输入的值 */
    printf("\n ***** 商品的详细信息 ***** \n");
    printf("商品编号:%s\n", goods1.id);
    printf("商品名称:%s\n", goods1.name);
    printf("单价:%5.2f\n", goods1.price);
    printf("数量:%d\n", goods1.number);
}

```

程序运行结果如图 lab. 8 所示。

【思考】

- (1) 在输入商品编号时能否输入 6 个字符?
- (2) 在输入商品编号或商品名称时字符串之间能否有空格?

练习 2: 使用结构数组, 在练习 1 的基础上编写一个程序, 统计商品总销售额。

【解题思路】

本例要求录入 N 条商品信息, 用“struct goods goods1[N];”语句定义商品结构数组, 引用第 i 种商品成员信息用 goods1[i].id、goods1[i].name、goods1[i].price 和 goods1[i].number 表示, 统计第 i 种商品销售额用 goods1[i].price * goods1[i].number 表示。

【参考代码】

```

#include <stdio.h>
#define N 2
struct goods
{ char id[6];
  char name[25];
  float price;
  int number;
};

```



图 lab. 8

```

void main()
{
    struct goods goods1[N];
    int i;
    float sum = 0;
    printf("**** 请输入商品的详细信息 **** \n");
    printf("编号 名称 单价 数量\n");
    for(i=0; i<N; i++)
    {
        scanf("%s", goods1[i].id);
        getchar(); //接收输入时用于商品编号与名称之间的回车符
        scanf("%s", goods1[i].name);
        scanf("%f", &goods1[i].price);
        scanf("%d", &goods1[i].number);
        sum += goods1[i].price * goods1[i].number;
    }
    printf("商品销售总额:%10.1f\n", sum);
}

```

程序运行结果如图 lab.9 所示。

练习 3: 结构变量作为函数参数。编写一个函数 fun(), 在练习 1 定义的结构基础上, 把结构变量作为函数参数传递, 求某种商品销售额。

【解题思路】

本题较简单, 只要阅读参考代码即可了解。

【参考代码】

```

#include <stdio.h>
struct goods
{
    char id[6];
    char name[25];
    float price;
    int number;
};
float fun(struct goods g1)
{
    return (g1.price * g1.number);
}
void main()
{
    struct goods goods1; // 结构变量
    printf("**** 请输入商品的详细信息 **** \n");
    printf("商品编号:");
    scanf("%s", goods1.id); //或用 gets(goods1.id);
}

```

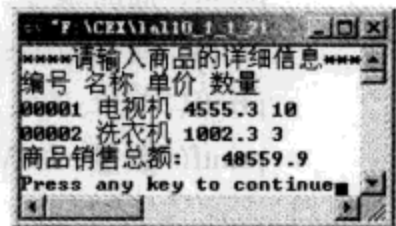


图 lab.9

```

getchar(); //用于接收商品编号与商品名称之间的回车符
printf("商品名称:");
scanf("%s", goods1.name); // gets(goods1.name);
printf("单价:");
scanf("%f", &goods1.price);
printf("数量:");
scanf("%d", &goods1.number);
/* 调用函数 fun()统计某种商品销售额 */
printf("商品销售额:%6.1f\n", fun(goods1));

```

程序运行结果如图 lab. 10 所示。

第二部分 练习

练习 4: 使用结构数组, 定义一结构 student, 其成员类型如表 lab. 2 所示。

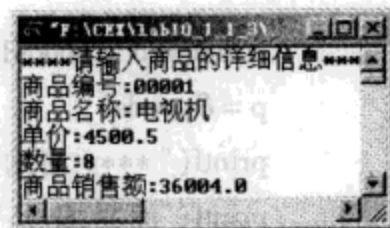


图 lab. 10

表 lab. 2

学号	姓名	数学
8 个字符	20 个字符	单精度浮点数

输入 N(N=3) 条学生信息, 采用结构数组方法(练习 2 的方法)计算学生的平均成绩, 并输出。

第三部分 作业

把练习 4 改造成用结构数组作为函数参数实现计算 N 名学生的平均成绩, 平均成绩作为返回值返回主调函数输出。

第四部分 思考、总结及书写实验报告

实验十四 结构与共用(二)

实验目的

- (1) 掌握指向结构变量的指针变量的定义与使用方法。
- (2) 掌握指向结构数组的指针变量的使用方法。
- (3) 理解结构变量的指针变量作为函数参数。

第一部分 教师指导

练习 1: 编写一个程序, 用于创建一个结构 goods, 用来存储商品信息: 商品编号、商品名称、单价和数量, 并用指向结构变量的指针变量来接收用户输入的值并显示这些值。

goods 结构成员类型如表 lab. 1 所示。

【解题思路】

结构的成员包括: 商品编号、商品名称、单价和数量。商品编号和商品名称用字符表示, 单价为浮点型, 数量是整型。

【参考代码】

```

#include <stdio.h>

struct goods
{
    char id[6];
    char name[25];
    float price;
    int number;
};

void main()
{
    struct goods goods1, *p;           //结构变量
    p = &goods1;                       //给指向结构变量的指针变量赋值
    printf("**** 请输入商品的详细信息 **** \n");
    printf("商品编号:");
    gets(p->id);                       //用此句下面的 fflush(stdin);就不需要了
    fflush(stdin);                     //清除键盘缓冲区信息
    printf("商品名称:");
    gets(p->name);
    printf("单价:");
    scanf("%f", &p->price);
    printf("数量:");
    scanf("%d", &p->number);
    printf("\n ***** 商品的详细信息 ***** \n");
    printf("商品编号:%s\n", p->id);
    printf("商品名称:%s\n", p->name);
    printf("单价:%5.2f\n", p->price);
    printf("数量:%d\n", p->number);
}

```

程序运行结果如图 lab. 8 所示。

【思考】

在输入商品编号时能否输入 6 个字符？

练习 2: 使用指向结构数组的指针变量, 在练习 1 的基础上编写一个程序, 统计商品总销售额。

【解题思路】

本例要求录入 N 条商品信息, 用“struct goods goods1[N];”语句定义商品结构数组, 给指向结构数组 goods[N] 的指针变量 p 赋初值为 goods1, 引用第 0 种商品成员信息用 p->id、p->name、p->price 和 p->number 表示, 统计第 0 种商品销售额用 p->price * p->number 表示, 然后 p++, 指向下一条数组元素。

【参考代码】

```

#include <stdio.h>
#define N 2

```

```

struct goods
{
    char id[6];
    char name[25];
    float price;
    int number;
};

void main()
{
    struct goods goods1[N], *p;
    float sum = 0;
    printf("**** 请输入商品的详细信息 **** \n");
    printf("编号 名称 单价 数量\n");
    for(p = goods1; p < goods1 + N; p++)
    {
        scanf("%s", p->id);
        getchar(); //接收输入时用于商品编号与名称之间的回车符
        scanf("%s", p->name);
        scanf("%f", &p->price);
        scanf("%d", &p->number);
        sum += p->price * p->number;
    }
    printf("商品销售总额:%10.1f\n", sum);
}
    
```

程序运行结果如图 lab. 9 所示。

练习 3: 编写一个函数 fun(), 在练习 1 定义的结构基础上, 把指向结构变量的指针变量作为函数参数传递, 求某种商品销售额。

【解题思路】

本题较简单, 只要阅读参考代码即可了解。

【参考代码】

```

#include <stdio.h>
#include <stdlib.h>
struct goods
{
    char id[6];
    char name[25];
    float price;
    int number;
};

float fun(struct goods *p1)
{
    return (p1->price * p1->number);
}
    
```

```

}
void main()
{
    struct goods *p;           // 结构变量
    p = (struct goods *) malloc( sizeof( struct goods ) );
    printf( "**** 请输入商品的详细信息 **** \n" );
    printf( "商品编号:" );
    scanf( "%s", p->id );       //或用 gets( p->id );
    getchar();                 //用于接收商品编号与商品名称之间的回车符
    printf( "商品名称:" );
    scanf( "%s", p->name );     // gets( p->name );
    printf( "单价:" );
    scanf( "%f", &p->price );
    printf( "数量:" );
    scanf( "%d", &p->number );
    /* 调用函数 fun() 统计某种商品销售额 */
    printf( "商品销售额:%6.1f\n", fun( p ) );
}

```

程序运行结果如图 lab. 10 所示。

第二部分 练习

练习 4: 使用指向结构数组的指针变量。定义一结构 student, 其成员类型如表 lab. 3 所示。

表 lab. 3

学号	姓名	数学
8 个字符	20 个字符	单精度浮点数

输入 N(N=3) 条学生信息, 采用指向结构数组的指针变量方法(练习 2 的方法) 计算学生的平均成绩并输出。

第三部分 作业

把练习 4 改造成用指向结构数组的指针变量作为函数参数, 实现计算 N 名学生的平均成绩, 平均成绩作为函数返回值返回主调函数输出。

第四部分 思考、总结及书写实验报告

实验十五 文件

实验目的

- (1) 掌握文件和文件指针的概念以及文件的定义方法。
- (2) 掌握打开和关闭文件的概念和方法。
- (3) 掌握有关文件操作的函数。

第一部分 教师指导

练习 1: 从键盘上输入一串字符, 把它输出到磁盘文件 test. txt 中, 并把该文件内容读出并

显示在屏幕上。

【解题思路】

本题是以字符形式进行文件的读写,所以用“w+”(“wt+”)方式打开文件,文件名为 test.txt。定义一个字符变量 ch 和字符数组 str,用于读/写一个字符或一个字符串。

几种字符文件操作函数如下。

(1) fputc(ch,fp) 写一字符到文件中, ch = fgetc(fp) 读一字符到 ch 中。

(2) fputs(str,fp) 写一字符串到文件中, fgets(str,size,fp) 读一大小为 size 字节的字符串到 str 中。

(3) fprintf(fp,"%s",str) 写一字符串到文件中, fscanf(fp,"%s",str) 读一个字符串到 str 中。

【参考代码】

```
#include <stdio.h>
#include <stdlib.h>
void main()
{ FILE *fp;
  char ch;
  if((fp = fopen("text.txt","w+")) == NULL) //判断文件能否打开
  {
    printf("不能打开文件:text.txt");
    exit(1);
  }
  printf("输入一串字符:\n");
  ch = getchar();
  /* 从键盘上读入字符,若不是回车符,则将字符写入文件,然后输入下一个字符 */
  while(ch != '\n')
  { fputc(ch,fp);
    ch = getchar();
  }
  rewind(fp); //将文件内部指针移动到文件头
  printf("刚输入的字符串:\n");
  ch = fgetc(fp);
  /* 从文件读入字符,若不是回车符,则将字符送屏幕显示,然后读入下一个字符 */
  while(ch != EOF)
  {
    printf("%c",ch);
    ch = fgetc(fp);
  }
  printf("\n");
  fclose(fp); //关闭文件
}
```

【思考】

(1) 请使用 fscanf() 和 fprintf() 函数从文件 test.txt 中读写一个字符串。

(2) 请使用 `fputs()` 和 `fgets()` 函数从文件 `test.txt` 中读写一个字符串。

程序运行的结果如图 lab. 11 所示。

第二部分 练习

练习 2: 从磁盘文件 `test.txt` 中读入一行字符到内存中, 将其中的大写字母改为小写字母, 显示在屏幕上, 然后再输出到 `test1.txt` 中。

【提示】

可用练习 1 的各种文件读写函数操作。

本题程序运行结果如图 lab. 12 所示。

第三部分 作业

有两个磁盘文件 `test.txt`、`test1.txt`, 各自存放若干字符, 要求将两个文件内容合并, 并将合并好的字符串输出到文件 `test2.dat` 中, 要求显示写入后的内容。

第四部分 思考、总结及书写实验报告

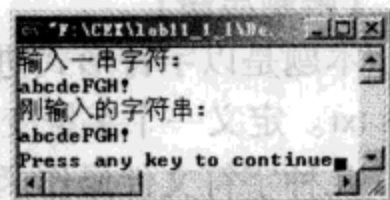


图 lab. 11

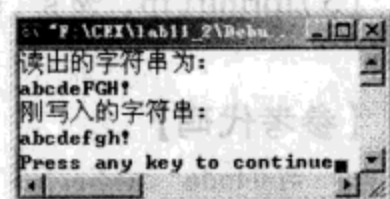


图 lab. 12

附录1 ASCII 表

ASCII 值	控制字符	ASCII 值	字符	ASCII 值	字符	ASCII 值	字符
0	NUL	32	(space)	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

附录 2 C 库函数所在头文件

1. 数学函数所在头文件“math.h”

主要数学函数如下: `abs(int x)` (取整数 x 绝对值)、`acos(double x)` (反余弦)、`asin(double x)` (反正弦)、`atan(double x)` (反正切)、`cos(double x)`、`cosh(double x)` (双曲余弦)、`exp(double x)` (e^x)、`fabs(double x)` (取 x 绝对值)、`floor(double x)` (不大于 x 的最大整数)、`fmod(double x, double y)` (求整除 x/y 双精度余数)、`log(double x)`、`log10(double x)`、`pow(double x)` (幂函数)、`sin(double x)`、`sinh(double x)` (双曲正弦)、`sqrt(double x)` (求 x 的平方根)、`tan(double x)` (正切)、`tanh(double x)` (双曲正切) 等。

2. 字符函数所在头文件“ctype.h”

主要字符函数如下: `isalnum(int ch)` (检查 ch 是否为字母或数字)、`isalpha(int ch)` (检查 ch 是否为字母)、`isdigit(int ch)` (检查 ch 是否为数字)、`islower(int ch)` (检查 ch 是否为小写字母)、`isupper(int ch)` (检查 ch 是否为大写字母)、`isspace(int ch)` (检查 ch 是否为空格、制表符或换行符)、`isxdigit(int ch)` (检查 ch 是否为十六进制数字)、`ispunct(int ch)` (检查 ch 是否为除空格、字母、数字以外可打印字符)、`tolower(int ch)` (转 ch 为小写字母)、`toupper(int ch)` (转 ch 为大写字母) 等。

3. 字符串函数所在头文件“string.h”

字符串函数如下: `strcat()` (连接)、`strcmp()` (比较)、`strcpy()` (复制)、`strlen()` (测字符串长度)、`strstr(s1, s2)` (在 $s1$ 串中找出 $s2$ 串第一次出现的位置) 等。

4. 输入/输出函数所在头文件“stdio.h”

主要输入/输出函数如下: `fclose(fp)` (关闭 fp 指的文件)、`feof(fp)` (检查文件是否结束)、`fgetc(fp)` (从 fp 所指文件中读取一个字符)、`fgets(fp)` (从 fp 所指文件中读取一个字符串)、`fopen(文件名, 打开方式)` (打开文件)、`fputc(fp)` (向 fp 所指文件中写一字符)、`fputs(fp)` (向 fp 所指文件中写一串字符)、`fread()` (读数据块)、`fwrite()` (写数据块)、`fprintf()` (向文件中以一定格式写信息)、`fscanf()` (从文件中以一定格式读信息)、`ftell(fp)` (求出 fp 所指文件当前的读写位置)、`fseek()` (移动文件内部的位置指针)、`rewind(fp)` (将文件内部位置指针移到文件首)、`getchar()` (从标准输入设备读取一个字符)、`gets()` (从标准输入设备读取一个字符串)、`putc()` (向标准输出设备输出一个字符)、`puts()` (向标准输出设备输出一个字符串)、`printf()` (向标准输出设备以一定格式显示信息)、`scanf()` (从标准输入设备以一定格式读入信息)。

5. 动态分配函数与随机函数的头文件“stdlib.h”

动态分配函数与随机函数如下: `calloc(n, size)` (分配 n 个数据项的内存空间, 每个数据项的大小为 $size$ 个字节)、`free(p)` (释放 p 所指的内存区)、`malloc(size)` (分配 $size$ 个字节的存储空间)、`realloc(p, size)` (把 p 所指的内存区的大小改为 $size$ 个字节)、`rand()` 产生 $0 \sim 32767$ 的随机整数。

参考文献

- [1] 谭浩强. C 语言程序设计[M]. 2 版. 北京:清华大学出版社,1999.
- [2] 谭浩强. C 语言程序设计题解与上机指导[M]. 北京:清华大学出版社,1995.
- [3] 孙辉. C 语言程序设计教程[M]. 北京:人民邮电出版社,2004.
- [4] 黄锐军. C 语言程序设计[M]. 北京:人民邮电出版社,2005.
- [5] 北大青鸟信息技术有限公司. 程序逻辑和 C 语言实现[M]. 北京:科学技术文献出版社,2005.
- [6] 郭嘉喜. C 语言程序设计教程[M]. 南京:南京大学出版社,2007.
- [7] 汪金泉. C 语言程序设计案例教程[M]. 北京:人民邮电出版社,2004.
- [8] 蒋清明. C 语言程序设计[M]. 北京:人民邮电出版社,2008.
- [9] 毕万新. C 语言程序设计[M]. 大连:大连理工大学出版社,2005.
- [10] 全国计算机等级考试新大纲命题研究组. 二级 C 语言程序设计[M]. 2 版. 北京:机械工业出版社,2006.
- [11] 陈华生. Visual C++ 程序设计基础[M]. 苏州:苏州大学出版社,2000.
- [12] 崔成. C 语言程序设计学习指导与习题汇编[M]. 北京:中国水利水电出版社,2007.
- [13] 宋铁桥. C 语言开发实例教程[M]. 北京:电子工业出版社,2008.
- [14] 石小玲. C 语言程序设计实例教程[M]. 北京:机械工业出版社,2004.
- [15] 李春葆. C 语言习题与解析[M]. 北京:清华大学出版社,2006.
- [16] 谢乐军. C 语言程序设计及应用[M]. 北京:冶金工业出版社,2004.
- [17] 高林. C 语言程序设计教程[M]. 北京:人民邮电出版社,2006.